Clube Delphi

Estrutura de Dados
Explore a utilização de listas

Git no XE7
A nova tendência em controle de versão

Edição 163 :: R$ 14,90

DEVMEDIA

SERVIÇOS NA NUVEM

Conheça o EMS

Sistema Financeiro
Centralização da conexão com o banco de dados

TMS TWebUpdate
Atualizações automáticas de forma fácil

MVP

R$ 1.000.000,00
INVESTIDOS EM CONTEÚDO NOS ÚLTIMOS 12 MESES.

APLIQUE ESSE INVESTIMENTO NA SUA CARREIRA...

E MOSTRE AO MERCADO QUANTO VOCÊ VALE!

CONFIRA TODO O MATERIAL QUE VOCÊ TERÁ ACESSO:

- + de **9.000** video-aulas
- + de **290** cursos online
- + de **13.000** artigos
- DEVMEDIA API's consumido + de **500.000** vezes

POR APENAS
R$ 69,90* mensais

*Tempo mínimo de assinatura: 12 meses.

ASSINATURA MVP MOST VALUABLE PROFESSIONAL

PRA QUEM QUER EXIGIR
MAIS DO MERCADO!

DEVMEDIA

Clube Delphi

163ª Edição 2015 ISSN 1517990-7 Impresso no Brasil

Editor Geral

Paulo Quicoli (pauloquicoli@gmail.com)

Equipe Editorial

Guinther Pauli (guintherpauli@gmail.com)

Fabrício Hissao Kawata (fabricio.kawata@bol.com.br)

Giuliano Scombatti Pinto (giuliano@sygnux.com.br)

Daniel Sobrinho Laporte (daniel.laporte@gmail.com)

Jornalista Responsável

Kaline Dolabella - JP24185

Consultor Técnico

Davi Costa (davigc_08@hotmail.com)

Capa e Diagramação

Romulo Araujo

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/central
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Paulo Quicoli - Editor da Revista
pauloquicoli@gmail.com

Paulo Quicoli - **Editor Geral**

pauloquicoli@gmail.com
quicoli.wordpress.com
twitter.com/pauloquicoli

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia: **www.devmedia.com.br/mvp**

DEVMEDIA

Sumário

04 – Utilizando Git no Delphi
[*Fabricio Hissao Kawata*]

Conteúdo sobre Novidade

14 – EMS: Conhecendo o Enterprise Mobility Services
[*Fabricio Hissao Kawata*]

Artigo no estilo Solução Completa

24 – Atualizando aplicações com TWebUpdate
[*Fabricio Hissao Kawata*]

Artigo no estilo Curso

36 – Desenvolvendo um Sistema Financeiro em Delphi – Parte 2
[*Filipe Dalepiane*]

Artigo no estilo Curso

42 – Estrutura de dados em Delphi – Parte 2
[*Filipe Dalepiane*]

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Utilizando Git no Delphi

Entenda os princípios deste novo tipo de sistema de controle de versão

A recente adição do sistema de controle de versão Git ao âmbito do Delphi acaba por agregar uma série de novas possibilidades ao próprio processo de desenvolvimento. A principal delas, sem sombra de dúvidas, se reflete numa forma mais controlada de trabalho em equipe, onde vários desenvolvedores estão inseridos, todos trabalhando de forma conjunta num mesmo projeto ou em atividades correlatas. Isso, em razão de suas próprias características, que provê, entre outras coisas, um ideal versionamento de código, o que acaba por manter um efetivo histórico de alterações a cada conteúdo, bem como uma prática solução para conflitos ocasionais (por exemplo, quando duas pessoas distintas alteram o conteúdo relacionado a um mesmo arquivo). Toda sua ação pode ainda ser superficialmente comparada ao Subversion (**BOX 1**), outro sistema de controle de versão bastante popular, cuja integração com o IDE já se encontra bastante estável uma vez que foi provida tempos atrás.

BOX 1. Subversion

Subversion é um sistema de controle de versão de código aberto (open source) distribuído de forma gratuita pela Apache, sob a tutela da Apache License. No popular, é comumente tratado como SVN e sua utilização é direcionada à manutenção de versões e históricos de arquivos, tais como os de código-fonte, de texto simples, documentos Web e outros tipos diversos. Num cenário recente, este sistema de controle de versão ganhou bastante notoriedade principalmente a partir de sua intensa adoção por parte da comunidade de software livre, o que inclui projetos já tradicionais como Mono, Free Pascal e FreeBSD.

Ainda sobre o Git, tendo em vista toda a extensão e complexidade natural do assunto, o presente artigo volta seu foco exclusivamente à integração do referido sistema de controle de versão com o IDE do Delphi XE7. Isso significa dizer que, neste primeiro momento, somente os pontos tidos como essenciais serão abordados, tornando o aprendizado mais facilitado e empolgante ao desenvolvedor.

Fique por Dentro

A nova versão do Delphi, a XE7, trouxe consigo uma série de novidades relacionadas aos mais diversos aspectos pertinentes ao desenvolvimento de software. Dentre essas está a integração nativa do IDE com o Git, que pode ser brevemente pontuado como sendo uma alternativa eficiente ao já popular Subversion, no que diz respeito ao versionamento de código. Em vista disso, o presente artigo busca esclarecer ao leitor, de forma ágil, o que vem a ser este novo sistema de controle de versão, explicitando suas características essenciais, incluindo todo seu aspecto prático de utilização no IDE. De forma objetiva, este é o ponto de partida para a plena introdução do Git no cotidiano de trabalho de todo desenvolvedor Delphi da atualidade.

Obviamente, a partir disso, o mesmo estará preparado para o passo seguinte, envolvendo aspectos mais complexos da conjuntura, os quais eventualmente serão abordados em artigos subsequentes. Por fim, toda a essência citada acaba por envolver então desde a definição conceitual do Git até seu efetivo uso por meio do IDE, em um contexto totalmente local (não remoto).

O que é um sistema de controle de versão e qual sua necessidade

Em vista do pleno entendimento e posterior utilização do Git, alguns conceitos iniciais devem ser assimilados pelos desenvolvedores. Dentre estes, o primeiro deles diz respeito à definição concreta do que vem a ser um sistema de controle de versão, que é o conceito ao qual o Git se enquadra. Sendo assim, de forma ampla, um version control system nada mais é do que um sistema capaz de persistir e manter as alterações (uma ou várias) realizadas em um determinado período de tempo sobre um único arquivo ou um conjunto deles. De posse dessas informações, esse tipo de sistema torna possível então a volta a momentos temporais específicos da vida do arquivo, que caracterizam então uma "versão" do mesmo.

Tendo em vista essa definição, especificamente no desenvolvimento de software, um sistema de controle de versão pode ser

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

entendido como o elemento central que irá monitorar os arquivos do projeto com relação a todo e qualquer tipo de mudança que venha a ocorrer sobre seu conteúdo. Já no lado prático, uma vez instalado, o sistema faz uso de um diretório local, comumente o mesmo lugar onde estão localizados os arquivos do projeto, para a gerência inicial de todo o histórico de alterações dos mesmos.

Além da definição conceitual, outro ponto relevante a ser compreendido com relação a um sistema de controle de versão diz respeito a sua necessidade de uso. Em outras palavras, a busca por uma resposta ideal para o seguinte questionamento: por que devo ou preciso utilizar um sistema de controle de versão? Para isto, alguns argumentos pontuais podem ser rapidamente utilizados:

• Automação de backups: um sistema de controle de versão é capaz por si só de automatizar toda a criação de "cópias de segurança" (backups) de arquivos. Logo, seu uso acaba por substituir qualquer processo manual semelhante, incluindo o já tradicional "copiar e colar – Ctrl+C/Ctrl+V" de diretórios de arquivos, onde cada cópia acaba por elucidar uma versão diferente dos mesmos (por exemplo, ArquivosDoProjeto, ArquivosDoProjeto-Old, ArquivosDoProjeto-Backup2011);

• Rastreabilidade de mudanças: um sistema de controle de versão acaba por dispor de forma direta uma rastreabilidade completa pelas mudanças ocorridas ao longo do tempo no conteúdo dos arquivos versionados. Num ambiente corporativo, isso vem a colaborar em questões envolvendo responsabilidades pessoais;

• Unicidade de nomenclatura: você poderá manter várias "versões" de um mesmo arquivo sob um único nome corrente (por exemplo, UCadClientes.pas). Isso evita diversos problemas, tais quais os transtornos causados pela tentativa de um controle manual semelhante, a partir do uso de nomenclaturas distintas a cada versão do mesmo arquivo (por exemplo, UCadClientes.pas, UCadClientes-old.pas, UCadClientes-ver1.pas);

• Antecipação de futuras alterações: você será capaz de marcar uma parte específica do conteúdo relacionado aos arquivos do projeto corrente a fim de implementar pontualidades futuras (por exemplo, novos recursos) sem que isso incida em qualquer tipo de mudança no conteúdo atual. Adicionalmente, a qualquer momento tais pontualidades poderão então ser integradas ao projeto em produção.

Em vista das funcionalidades que um sistema de controle de versão pretende oferecer a seus utilizadores, seu uso deve ser o mais simplificado possível, a fim de justificar sua adoção. Do contrário, um alto nível de complexidade acabaria por resultar em uma provável degradação e rejeição por parte das pessoas. Por esta razão, a inserção do Git em qualquer cenário cotidiano de trabalho é comprovadamente simples e funcional, conforme será visto no decorrer deste artigo.

Tipos de sistemas de controle de versão

Dada sua relativa complexidade e ampla diversidade, os sistemas de controle de versão são segmentados em tipos bem definidos, mais especificamente três, em que cada qual é classificado de acordo com seu modo de operação – local, centralizado ou distribuído.

Sistema de controle de versão local

O tipo local pode ser estabelecido como sendo a primeira alternativa surgida como forma de solução da questão envolvendo o versionamento manual por meio da nomenclatura de arquivos, que até então se mostrava como uma atividade muito propensa a erros. Historicamente, todo este cenário imposto por este tipo "local" pode ser traduzido numa ferramenta surgida ainda em meados da década de 80, denominada Revision Control System (ou Sistema de Controle de Revisão, numa tradução livre para o português). Como característica, o RCS mantinha todo o rastro progressivo de versões dos arquivos de forma local, ou seja, no próprio disco rígido operante, utilizando para isso um formato especial próprio. Do lado técnico, a essência básica de sua funcionalidade envolvia a manutenção de conjuntos de patches, em que cada qual carregava as diferenças existentes entre os diversos estágios de evolução do arquivo. Consequentemente, a restauração de um arquivo em determinado ponto de seu ciclo de vida acabava por significar a junção de vários patches ordenados até ali.

Sistema de controle de versão centralizado

Seguindo por essa natural evolução dos sistemas de controle de versão, mediante as novas necessidades surgidas por parte de seus usuários, um novo tipo foi sendo introduzido no processo, agora marcado pela centralização do mesmo. A maior justificativa para isso se deu em razão da dificuldade colaborativa imposta por um controle local, o que impedia o pleno trabalho em equipe num mesmo conjunto de arquivos (projeto). Diante desse dilema, a solução veio então por meio da introdução de um servidor, que caracterizava então um lugar comum, onde todos os envolvidos no processo colaborativo pudessem ter acesso a partir de suas estações locais. Esta nova ambientação deu origem então a uma plena relação entre servidor e clientes, que é a característica essencial de um sistema de controle de versão centralizado. Já pelo lado prático, de forma fundamental, algumas especificidades são providas por este modelo:

• Edição de arquivos: para a efetiva manipulação de um arquivo já "versionado", somente sua última versão é recuperada do servidor. Em vista da forma de trabalho proposta por este tipo de sistema, isso é mais que suficiente às suas pretensões;

• Visibilidade de trabalho: a configuração deste modelo centralizado provê não somente acesso aos arquivos para os envolvidos interessados, como também oferece formas de visibilidade dos trabalhos desenvolvidos por cada um;

• Reflexo global de mudanças: neste modelo, pelo fato dos arquivos versionados ficarem armazenados em um local único e centralizado, quaisquer alterações realizadas sobre eles serão automaticamente refletidas e compartilhadas a todas às pessoas envolvidas.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Sistema de controle de versão distribuído

Novamente seguindo por uma evolução natural dos processos envolvidos aos sistemas de controle de versão, com o passar do tempo, algumas adversidades acabaram sendo notadas mediante o uso do modelo centralizado. Dentre estas, é apontada uma questão de segurança, justificada em razão dos usuários de sistemas de controle de versão centralizado só trabalharem habitualmente com a última versão dos arquivos. A partir disso, uma eventual falha no servidor, que resulte na corrupção de seus dados, faz com que todo o histórico de versões seja perdido, ficando o usuário com a posse somente da última versão dos arquivos obtidos.

A busca por uma solução plenamente eficaz para este novo "problema" surgido acabou por resultar na combinação dos dois modelos existentes até então, o local e o centralizado, o que deu origem a um novo tipo de sistema de controle de versão denominada "sistema de controle de versão distribuído" (Distributed Version Control System). Em suma, foi feita então uma mescla das vantagens proporcionadas por cada tipo:

- Todo trabalho realizado ocorre a partir da posse de uma cópia local dos arquivos mantidos no servidor, o que garante uma total independência do mesmo;
- Logo, as atividades envolvendo as alterações de conteúdo dos arquivos ocorrem de maneira "local", sem qualquer envolvimento online com o servidor;
- A existência de um servidor é o suficiente por garantir todo o aspecto do trabalho colaborativo desejado, incluindo reusabilidade de trabalhos.

Funcionalmente, um sistema de controle de versão distribuído é projetado então para armazenar todo o histórico de mudanças dos arquivos de forma local. Ao mesmo tempo, sincroniza toda e qualquer alteração local enviada de volta ao servidor, fazendo com que essas sejam devidamente compartilhadas com as outras pessoas envolvidas no ambiente de trabalho colaborativo.

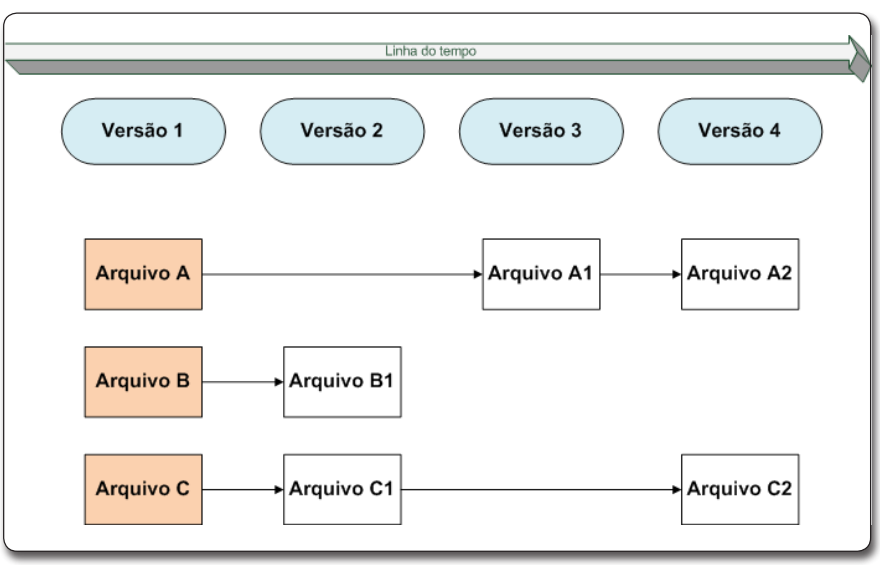

Figura 1. Mecânica de versionamento clássico

Git

Por definição, o Git é um dentre os vários sistemas de controle de versão distribuído presentes no mercado. Historicamente seu primeiro protótipo concreto foi idealizado e disponibilizado em meados de 2005 podendo ser considerado, portanto, uma tecnologia recente. Desde o início de seu projeto, sob a atenção de Linus Torvalds, o famoso idealizador do kernel do sistema operacional Linux, o Git foi projetado para ser muito melhor e menos suscetível a falhas do que qualquer outro sistema de controle de versão existente. Engajamentos e idealismos a parte, fato é que o Git acaba por apresentar uma arquitetura comprovadamente rápida e flexível o suficiente para otimizar sua usabilidade. Seguindo por esta linha, três quesitos se tornam fundamentais para sua definição: performance, segurança e atomicidade.

Performance

Mediante sua forma de trabalho, o Git pode ser considerado mais rápido que seus concorrentes. Isso se deve muito ao fato de que muitas de suas principais operações, efetivadas de forma remota em outros sistemas de controle de versão, se dão de forma local em seu ciclo processual. Logo, cada operação desta é sempre concluída em poucos segundos, o que acaba por superar as eventuais diversidades ocasionadas num cenário cliente-servidor.

Outro ponto de incidência direta sobre a questão da performance está relacionado à própria forma de interpretação do versionamento ocasionado. Isto porque, em sistemas como o Subversion uma "versão" nada mais é do que uma composição baseada em um conjunto definido de arquivos, acompanhad o das alterações realizadas sobre cada um deles. Pela ilustração mostrada na **Figura 1**, nota-se então os espaços vazios entre uma versão e outra, na trajetória linear do ciclo de vida de um arquivo. De forma fundamental, nos bastidores, isso tende a consumir performance do mecanismo do sistema, uma vez que toda uma análise deve ser feita para a composição final da exata versão requerida pelo usuário.

Por outro lado, o Git, ao invés de armazenar linearmente as diferenças entre uma versão e outra, acaba por tirar uma cópia instantânea (snapshot) de todo o conjunto de arquivos condizentes a cada nova versão criada (**Figura 2**). Em termos práticos, isso resulta em uma recuperação de versão mais rápida, sempre que requerida, uma vez que nenhuma análise interna se faz necessária em razão da plena disponibilidade de todos os arquivos que compõem uma determinada versão. Além disso, vale ressaltar que mesmo pelo fato do mecanismo do Git operar com snapshots, ele não trabalha com "cópias burras" de todo e qualquer arquivo. Ao contrário, em arquivos onde nenhuma mudança de conteúdo é detectada, o sistema acaba por guardar apenas uma referência ao snapshot anteriormente existente.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Segurança

A questão "segurança" também pode ser considerada um diferencial em meio às características do Git, principalmente no que diz respeito à integridade do conteúdo dos arquivos que estão sob o seu domínio. Isso porque tudo que vai para o sistema é verificado e sumarizado pelo uso de uma chave hash (SHA-1) antes de ser efetivamente armazenado. Ainda sobre esta chave, ela é composta por quatro dezenas de caracteres hexadecimais, gerada com base no conteúdo estrutural do arquivo ou diretório. A partir disso, reduz-se a quase zero a possibilidade do conteúdo de qualquer arquivo versionado ser alterado por um agente externo (humano ou não), sem que o núcleo do Git tome conhecimento.

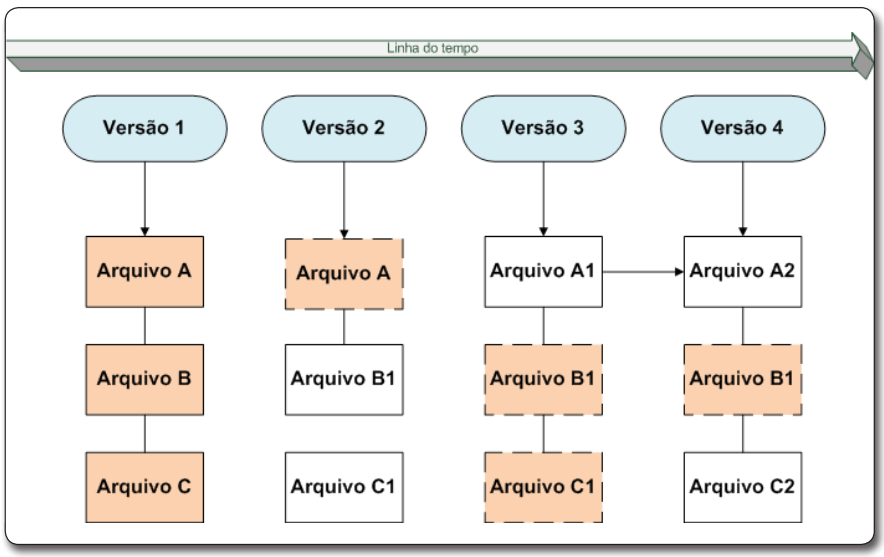

Figura 2. Mecânica de versionamento com Git

Atomicidade

Aqui, o termo atomicidade refere-se à propriedade de uma operação que ocorre essencialmente em um único instante, desde sua invocação até sua resposta (retorno). No popular, é tido como o famoso "tudo ou nada", em que uma vez que a operação é iniciada (por exemplo, saque bancário), a mesma irá falhar por completo ou obter o êxito esperado. Assim, nenhum resultado parcial é permitido. Com o Git, essa atomicidade é então imposta a todas as suas principais atividades, o que garante que não haja perda de dados ou eventuais incompatibilidades causadas por operações parciais. Todo esse aspecto, idealizado por seus criadores, garante ao sistema um nível de confiabilidade e estabilidade muito satisfatório a seus usuários.

Características comerciais

Do ponto de vista comercial, assim como muitos de seus concorrentes, o Git é disponibilizado de forma gratuita na Web (ver endereço na seção **Links**). Isso se deve muito ao fato do projeto do sistema ser de código aberto (open source), o que gera um movimento colaborativo muito grande em torno de seu próprio desenvolvimento. Prova disso é a variedade de plataformas suportadas atualmente pelo produto, mais especificamente quatro, todas elas de características bastante distintas:

- Microsoft Windows;
- Linux;
- Mac OS X;
- Solaris.

Download e instalação

Mediante o contexto do novo Delphi XE7, o primeiro passo para o uso efetivo do Git se dá por meio de seu download e subsequente instalação. Isto porque, diferente do Subversion, neste momento o utilitário não acompanha o setup padrão do IDE, sendo provida apenas sua integração nativa. Como fator positivo desde tipo de abordagem está o fato do desenvolvedor ter uma liberdade maior em escolher a versão do software que seja de seu agrado. Não que com o SVN tal ponto seja uma restrição, porém aqui, torna-se um fator explícito. Logo, a obtenção definitiva do Git é feita através da própria página oficial do projeto, cujo link é disponibilizado na seção **Links** ao final do artigo. Obviamente, em razão do Delphi ser um produto exclusivo para o sistema operacional Windows, esta deverá ser a plataforma escolhida durante a seleção da correta distribuição do Git (**Figura 3**).

Figura 3. Git – Plataformas suportadas e disponíveis

No geral, apesar de seguir por um processo bastante tradicional, a ideal instalação do Git requer pequenos ajustes que fogem de suas marcações padrão, visando sua plena utilização não só em razão do Delphi, mas também por eventuais ocasionalidades externas ao IDE. Assim, a primeira dessas particularidades diz respeito à sua etapa de seleção de componentes, mais especificamente os relacionados à integração com o Windows Explorer, onde as opções Git Bash Here e Git Bash GUI Here (**Figura 4**) devem ser marcadas, numa situação bastante autoexplicativa.

Seguindo por esta linha, outro ajuste necessário é ilustrado na **Figura 5**, onde é então selecionada a opção "Use Git from Windows Commando Prompt" que, conforme sua própria denominação sugere, habilita o usuário a trabalhar com o command-line do próprio SO para operar o Git. Do ponto de vista mais funcional, tais modificações são suficientes para as atividades do cotidiano.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

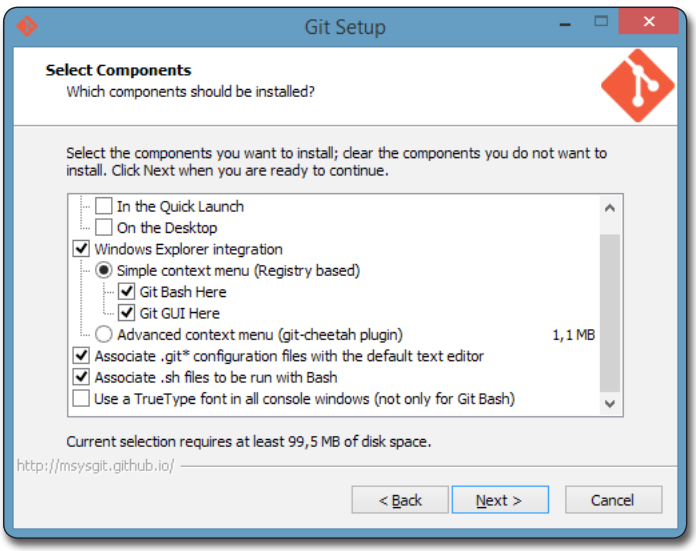

Figura 4. Seleção de componentes – Git Setup

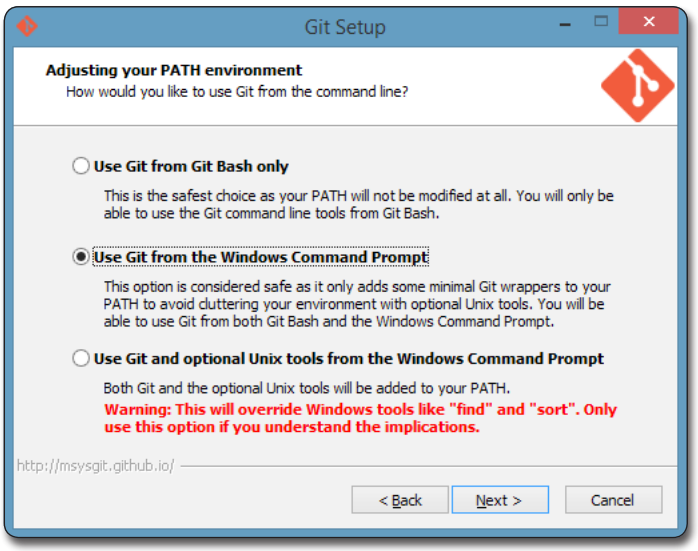

Figura 5. Git – Git Setup

Git no XE7 – Principais operações

A etapa imediatamente anterior à utilização efetiva do Git e sua integração com o IDE do Delphi XE7 se reflete no devido conhecimento das operações disponíveis do sistema para uso. Neste contexto, elas são basicamente duas, conforme descritas a seguir:

• Commit: operação do sistema de controle de versão que fundamentalmente envia as eventuais mudanças/alterações realizadas no conteúdo dos arquivos que estão sendo trabalhados, para o repositório local;

• Clone: operação do sistema de controle de versão que efetua uma cópia exata (clone) e local de um repositório remoto que esteja sob o domínio do Git.

Criando um Repositório

A utilização do Git por meio do IDE do Delphi pressupõe ainda a mínima existência de um repositório remoto, que pode ser considerado então o ponto de partida das atividades relacionadas ao versionamento de código-fonte de um projeto. Logo, a criação deste independe da ferramenta de desenvolvimento em si, ficando a cargo então de um processo manual mais tradicional. Neste ponto, entra em ação a ferramenta utilitária Git Gui, instalada junto ao sistema de controle de versão de forma nativa. Dessa forma, ao acioná-la, sua interface é mostrada (**Figura 6**), já com a opção para criação de um novo respositório disponível. Obviamente esta mesma abordagem poderia ser feita de um modo ainda mais manual, através de instruções de linha de comando, opção esta que fica a critério do desenvolvedor.

Ao acionar a referida opção, seja por meio da opção "Criar novo repositório" ou do atalho Ctrl + N, o Git Gui solicita então o caminho de um diretório válido, representativo do repositório raiz. Aqui, interprete este termo como sendo o "repositório remoto" do contexto, ou seja, o local onde no final das contas estarão alocados essencialmente todos os arquivos de projetos versionados de forma definitiva (não-local). Em se tratando de uma exemplificação local, aqui, o seguinte caminho será pautado, de forma a elucidar um cenário remoto (da presença de um servidor exclusivo):

```
C:/CD/Server/Exemplo
```

A partir deste ponto, o Git passa então a "monitorar" este diretório à espera de conteúdo.

Figura 6. Git Gui

Configurações iniciais

A configuração global do Git acaba por envolver uma série de fatores que por vezes fogem de uma abordagem inicial. Muitos destes pontos tendem a fomentar situações mais avançadas, que podem ser tidas como "não essenciais". Em vista disso, aqui serão pautadas somente aquelas caracterizadas como extremamente fundamentais para uso efetivo do sistema de controle de versão em questão. Dentre essas, a mais emergente, após a criação de um repositório inicial, está relacionada à criação da identidade (usuário) padrão que será utilizada para se registrar o log de alterações submetidos ao Git. Para tal, uma alternativa mais ágil e simplista se faz pelo uso da própria interface do Git Gui, mais

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

especificamente sua tela de opções, acessível por meio de seu menu Editar. A **Figura 7** ilustra então este cenário, já apresentando em destaque o devido preenchimento dos campos "nome do usuário" e "endereço de e-mail", relacionados tanto ao repositório local recém-criado, quanto de forma global (o que inclui todos os repositórios eventualmente existentes, atrelados a esta mesma instalação).

Vale ressaltar também que neste momento o Git já identifica o repositório criado, tal como pode ser visto na denominação utilizada para a coluna da esquerda (Repositório Exemplo, onde "Exemplo" é o nome do diretório anteriormente criado). Por fim, apenas isso é o suficiente para a efetiva utilização de todo o poder de gerenciamento de versão proporcionado pelo Git.

Sem deixar de citar, assim como a grande maioria das ações e operações desempenhadas pelo sistema, esta mesma configuração pode ainda ser realizada por meio do uso de linhas de comando. Inclusive, tal abordagem é a sugerida pela Embarcadero, em suas orientações oficiais relacionadas ao produto. Sendo assim, a seguir são explicitadas as sintaxes padrão dos devidos comandos envolvidos, conforme a **Listagem 1**.

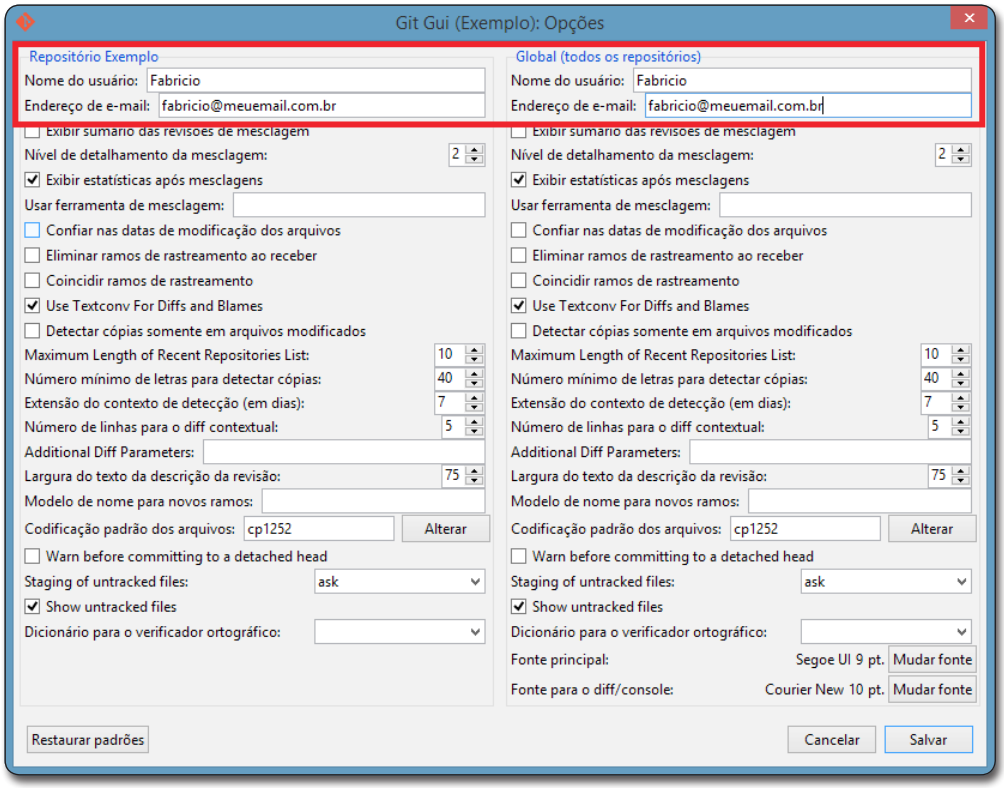

Figura 7. Git Gui – Tela de opções

Em vista das diversas palavras-chave utilizadas nas instruções, na sequência é provida uma breve descrição explicativa sobre cada uma:

- git: indica o uso do utilitário de linha de comando do sistema de controle de versão;
- config: indica o manuseio de determinada configuração do sistema;
- --local: indica a atribuição de um valor para um parâmetro de configuração local;
- --global: indica a atribuição de um valor para um parâmetro de configuração geral (global);
- user.name: aponta a atribuição direta ao parâmetro "nome" do usuário;
- user.email: aponta a atribuição direta ao parâmetro "email" do usuário.

De forma complementar, pelo fato da abordagem via linha de comando não contar com nenhuma segmentação visual, a checagem de confirmação das configurações realizadas ocorre por meio do seguinte comando:

```
git config --list
```

Por conseguinte, na **Listagem 2** a seguir são mostradas as instruções já adequadas à exemplificação utilizada na **Figura 7.**

Listagem 1. Comandos Git

```
//Comando para a configuração local do nome do usuário
git config --local user.name "seu nome"

//Comando para a configuração global do nome do usuário
git config --global user.name "seu nome"

//Comando para a configuração local do endereço de e-mail do usuário
git config --local user.email "seu email"

//Comando para a configuração global do endereço de e-mail do usuário
git config --global user.email "seu email"
```

Listagem 2. Instruções globais e locais

```
//Configuração local
git config --local user.name "fabricio"
git config --local user.email "fabricio@meuemail.com"

//Configuração global
git config --global user.name " fabricio "
git config --global user.email "fabricio@meuemail.com"
```

Adicionando arquivos no repositório

No contexto do Delphi, o ponto inicial do processo de versionamento com o Git se dá pela plena configuração do ambiente,

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

bem como a pré-existência de um repositório ativo. Para este último, isso acaba por incluir o carregamento inicial dos arquivos a serem versionados, simulando uma situação real onde o IDE acaba por atuar apenas como um simples cliente de um servidor já estabelecido. Numa situação mais comum, a simples cópia dos arquivos iniciais para o diretório do repositório é o suficiente para que os mesmos fiquem sob o contexto do Git. Tomando como base um projeto Delphi simples, do tipo VCL Forms Application, a **Figura 8** ilustra o cenário citado.

Figura 8. Arquivos no diretório do repositório

A partir disso, de forma automática, o próprio sistema de controle de versão já toma ciência da existência de tais arquivos. Isso é comprovado na própria interface do Git Gui, conforme pode ser visto na **Figura 9**. Seguindo por esta linha, a ação seguinte é fazer com que o Git passe a efetivamente interpretar tais arquivos. Neste layout, isso é conseguido através de um duplo-clique sobre os ícones de cada arquivo, o que resulta na passagem dos mesmos para o painel "Mudanças marcadas" Staged changes, como mostra a **Figura 10**.

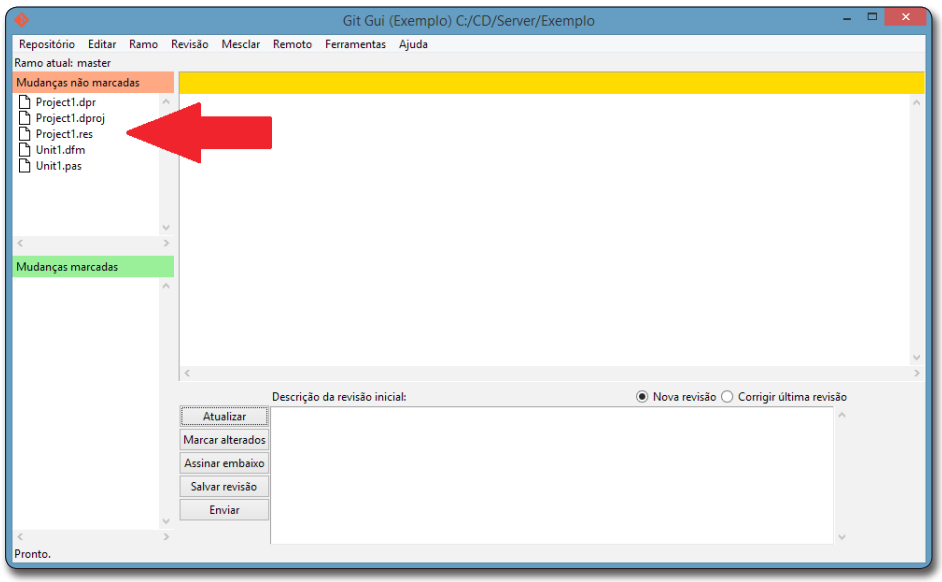

Figura 9. Git Gui – Arquivos iniciais do repositório

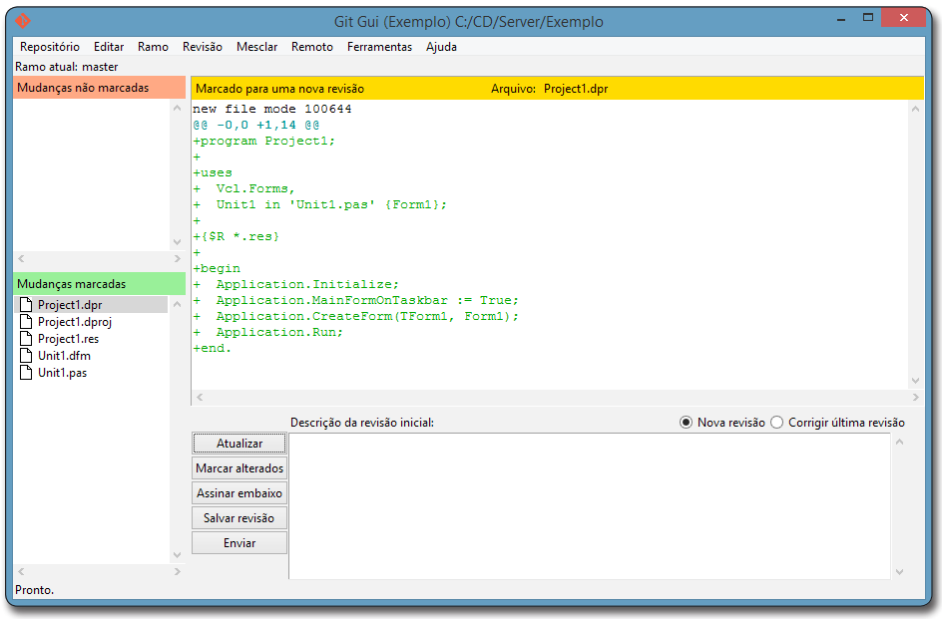

Figura 10. Git Gui – Painel "Mudanças Marcadas"

Commit inicial / Primeiro Clone

No Delphi, desde o surgimento de sua integração nativa com o Subversion, seu IDE passou a contar com uma opção exclusiva para a abertura de um projeto proveniente de um sistema de controle de versão, denominada "Open From Version Control...", disponível no menu File do próprio ambiente. Agora no XE7, ao acionar tal opção, uma nova caixa de diálogo é apresentada ao usuário, para a correta seleção do sistema de controle de versão em questão, em vista das duas possibilidades predominantes no momento (**Figura 11**) – Subversion e Git. Posteriormente, a correta indicação dos diretórios de origem (repositório) e destino (pasta de trabalho local), tal como mostra a **Figura 12**, é o suficiente para que o projeto seja recuperado a partir do repositório de versionamento para um diretório local e seja então imediatamente aberto no IDE. Pela imagem, de acordo com o Caption da janela, é possível destacar ainda a ação indicativa de "Clone", que reflete a uma das operações básicas do Git.

Neste momento, ainda que os arquivos do projeto já estejam no diretório do "repositório remoto", os mesmos ainda não podem ser considerados como estando sob um controle de versão, tal como comprova a **Figura 13**, cuja mensagem indicativa faz menção a um repositório ainda vazio – "You appear to have cloned an empty repository". Isso ocorre, porque perante o Git, todo e qualquer arquivo só é considerado "versionado" após o seu efetivo commit. Logo, para tal, ainda na interface do Git Gui, o Commit ocorre por

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

meio do botão "Salvar Revisão". Traçando um paralelo com o Subversion, aqui o processo commit é também acompanhado de um comentário descritivo, onde devem estar explicitados os detalhes pertinentes a esta etapa. Por se tratar de uma revisão inicial, a própria interface do Git já identifica o fato, apresentando a indicação "Descrição da revisão inicial", conforme pode ser visto na **Figura 14**. Vale ressaltar que todo este processo de commit visto neste momento, trata-se apenas de um ilustrativo de exemplo para a carga inicial de arquivos versionados no repositório tido como "remoto" nesta abordagem.

Abrindo um projeto versionado no Git

Uma vez que os arquivos estejam devidamente comitados e versionados, a abertura do projeto no IDE novamente se dá pela opção relacionada (Open From Version Control), porém agora de maneira fluente. Isso acaba por incidir em uma nova janela, posterior à clonagem do repositório remoto, que explicita ao desenvolvedor todos os arquivos encontrados válidos e indicativos de um projeto Delphi. Em outras palavras, todos os arquivos com a extensão .dproj, tal como visto na **Figura 15**.

Nos bastidores, essa ação ocorre imediatamente após a clonagem do repositório remoto para o diretório local apontado. Assim, localmente, já de posse dos arquivos do projeto, o Delphi é capaz de fazer a referida checagem. Ainda sobre a ação de Clone do Git no IDE, que marca o início da abertura de um projeto versionado, esta só ocorrerá caso o repositório envolvido não opere com autenticação. Isto porque, até o presente momento, o IDE não provê suporte ao provimento de dados de autenticação no momento da ação de Clone. Para estes casos, todo o processo citado, incluindo a autenticação, deve ser feito fora do IDE, seja pelo uso de linhas de comando ou mesmo pelo utilitário do próprio Git. Sem deixar de citar, o cenário mais clássico para uso de autenticação se dá nos casos de uso de repositórios remotos reais, uma vez que estes comumente tendem a integrar todo um aspecto de segurança, o que acaba por envolver o uso de credenciais de acesso (usuário e senha).

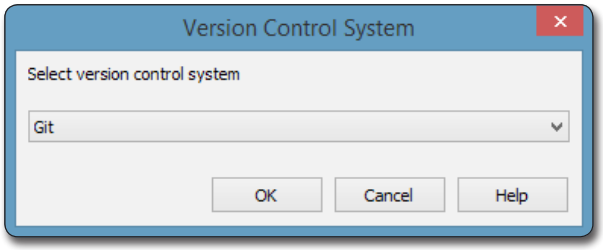

Figura 11. Janela de seleção do sistema de controle de versão

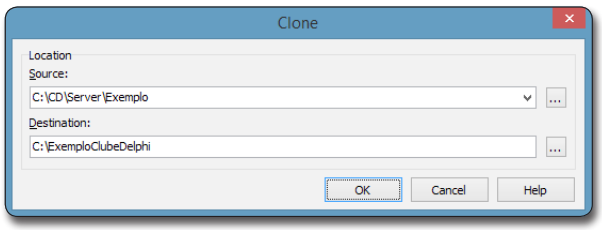

Figura 12. Janela de Clone

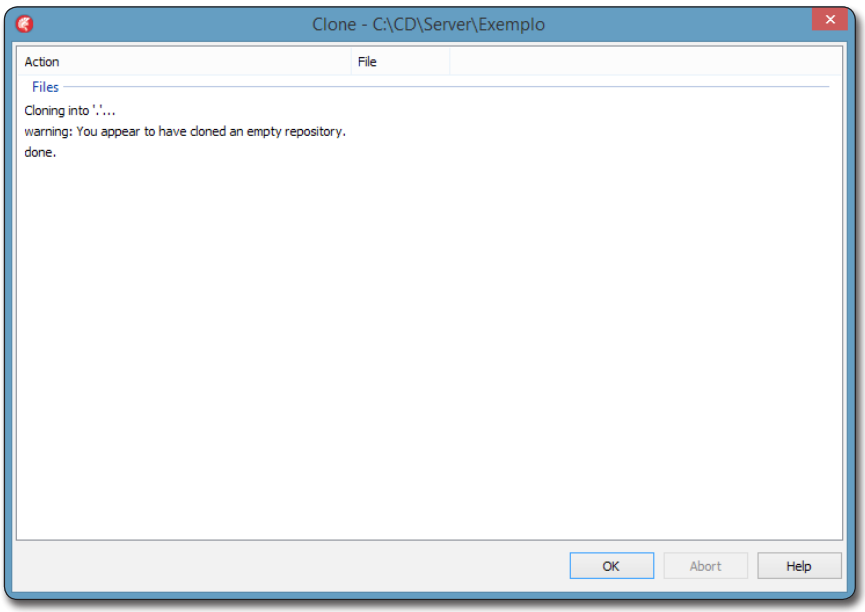

Figura 13. Janela de Clone – Arquivos

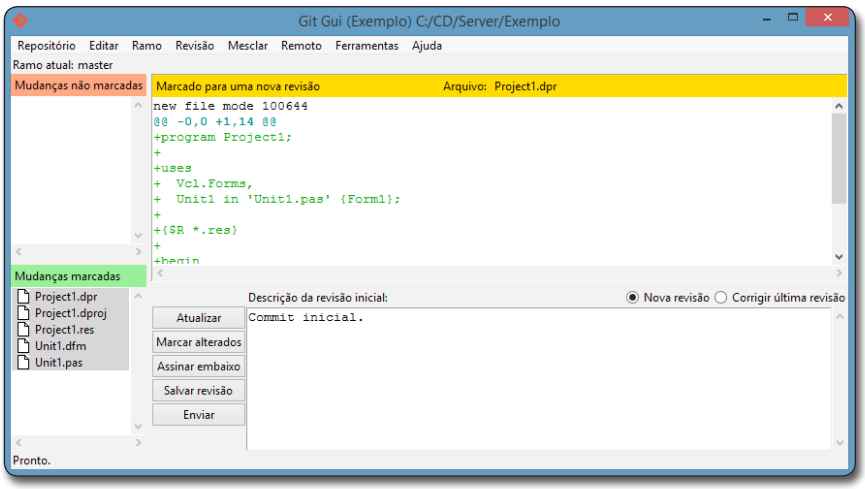

Figura 14. Commit inicial

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Commit para o repositório local

No Delphi, de posse do projeto versionado, qualquer eventual alteração no conteúdo de seus arquivos já habilita a ação de Commit no IDE. Nesta etapa, é importante ter em mente que, diferente do Subversion, tal ação envia as mudanças apenas para o repositório local. Na prática, isso se dá por meio do próprio Code Editor, mais especificamente num item homônimo disponibilizado na opção Version Control de seu menu de contexto (**Figura 16**).

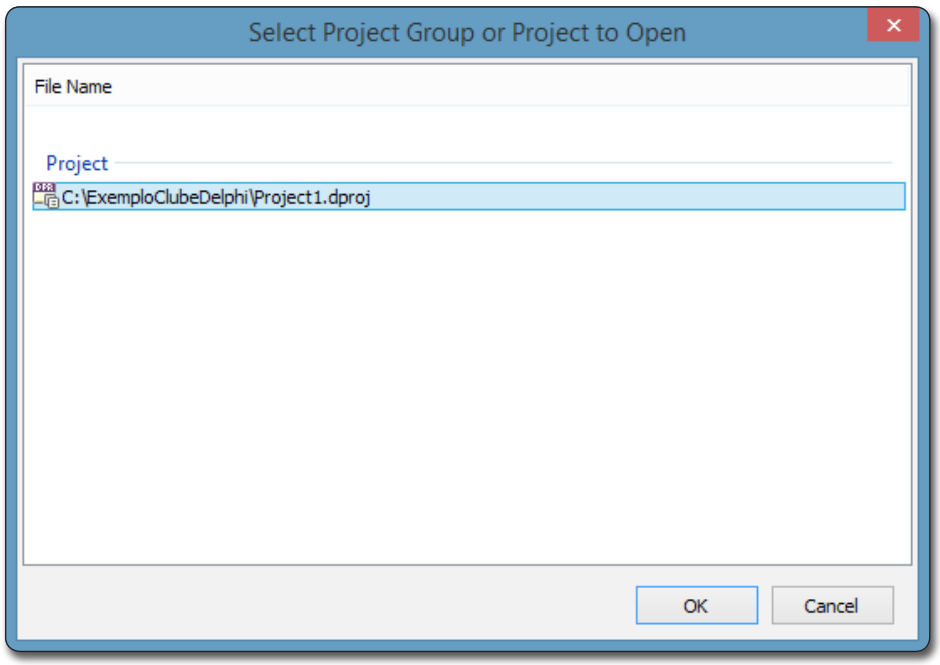

Figura 15. Arquivo de projeto provindo do sistema de controle de versão

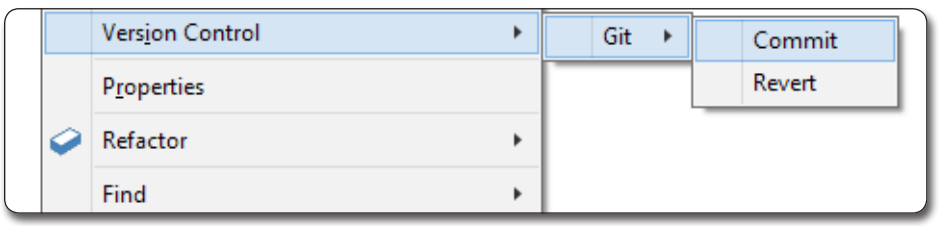

Figura 16. Git – Opção de Commit no IDE

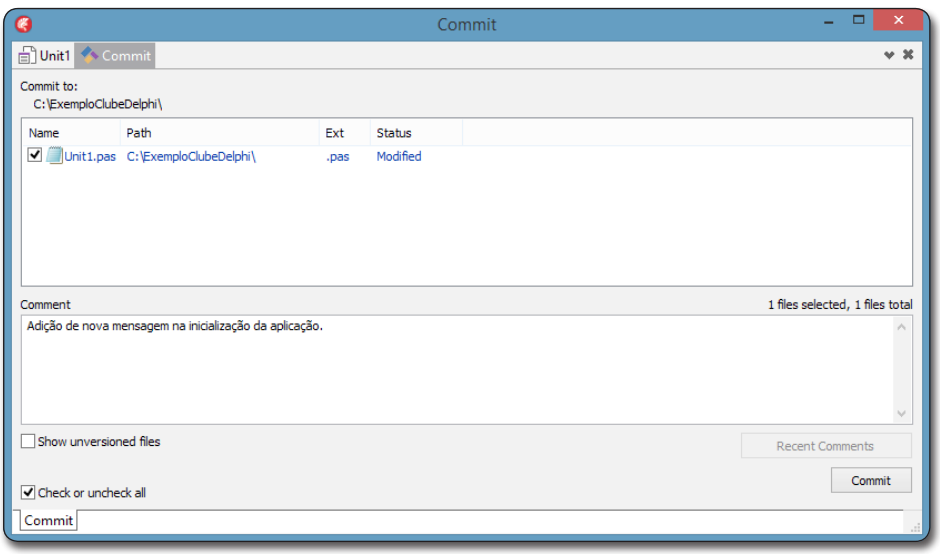

Figura 17. Git – Janela de Commit

Ao acionar esta ação, uma janela específica é então aberta, onde os detalhes do Commit podem ser previamente conferidos, o que inclui essencialmente os arquivos envolvidos. Adicionalmente, é provido também um campo informativo para a inclusão dos devidos comentários que irão fundamentar esta nova revisão. Todos estes componentes podem ser vistos na **Figura 17**. De forma breve, a função de cada um deles é listada a seguir:

- Indicativo "Commit to": indica o local para onde será realizado o commit. No Git, em se tratando de uma ação local, neste caso, ela é direcionada para o mesmo diretório do projeto, uma vez que este representa o próprio repositório local;
- Grid (Name, Path, Ext, Status): faz referência ao nome, localização, extensão e status do(s) arquivo(s) envolvido(s). Para este último Modified refere-se a um conteúdo que foi modificado, enquanto Added, um novo conteúdo;
- Caixa de texto "Comment": área digitável que permite a inserção de comentários pertinentes ao processo que está sendo realizado. Em ambientes de desenvolvimento, dois exemplos bastante comuns presentes neste tipo conteúdo se fazem pelo identificador (ID) do problema (bug) corrigido ou mesmo a correta indicação técnica de toda a atividade realizada;
- Caixa de marcação "Show unversioned files": mostra ou oculta os arquivos do projeto que não estão sob o controle de versão em questão;
- Caixa de marcação "Check or uncheck all": utilitário que marca ou desmarca todos os arquivos listados no Grid;
- Botão "Recent Comments": conforme seu indicativo, exibe uma janela contendo os comentários mais recentemente utilizados;
- Botão "Commit": efetivamente realiza o commit para o repositório local de todos os arquivos selecionados na grade.

Seguindo pelo paradigma do Git, como tendência, a ação posterior a um Commit é denominada Push que, tal como indica sua

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

nomenclatura, terá a incumbência de "empurrar" as mudanças realizadas de forma local de volta ao repositório remoto. Novamente, a versão corrente do IDE não provê suporte a tal funcionalidade, sendo necessária executá-la externamente, como por exemplo, por meio do comando git push.

Visualizando o Log de versionamento

Continuando o exemplo sobre a integração Delphi-Git, outro recurso provido pelo IDE é a possibilidade de visualização do log do controle de versão. Assim como a grande maioria das funcionalidades disponíveis na ferramenta, tal recurso é acionado de maneira totalmente visual por meio do menu de contexto do próprio projeto, presente no Project Manager (Git > Show Log). Logo, uma nova janela é aberta, tal como mostrada na **Figura 18**. Tendo em vista que o Git trabalha com cópias de repositórios, é fato que cada um destes poderá conter vários diretórios associados e, consequentemente, possuir vários projetos distintos. Por esta razão, ao acionar a opção Show Log indicada, dois caminhos singulares são providos ao desenvolvedor, com relação à origem do Log:

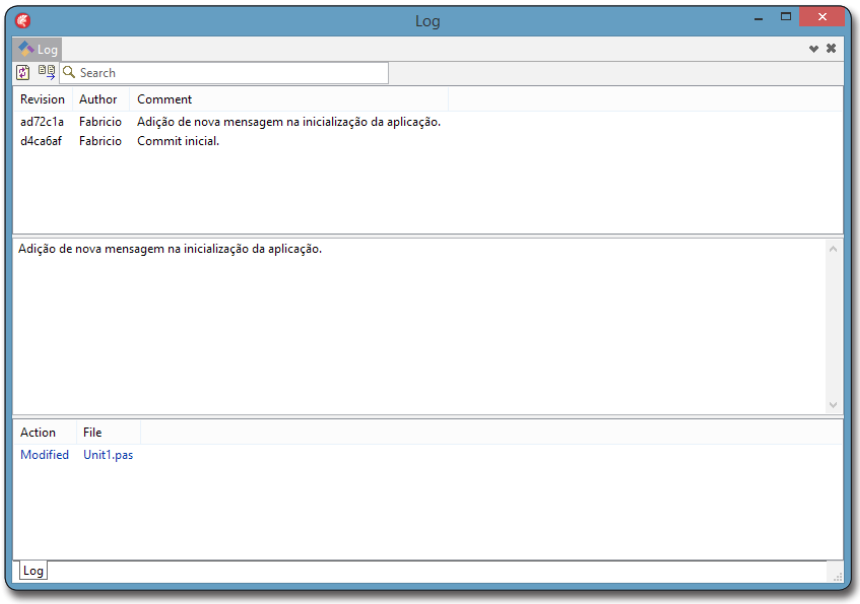

Figura 18. Git – Janela de Log

- From Repository Root: exibe o Log do último diretório gerenciado pelo mesmo repositório que o arquivo de projeto corrente;
- Botão "Recent Comments": exibe o Log de todos os arquivos localizados no diretório do projeto.

Outros comandos suportados

Na prática, cada operação disponibilizada no IDE e que esteja relacionada ao Git, nos bastidores é acionada por meio de um comando Git padrão. Como exemplo, conforme já visto, o acionamento da opção "Open from Version Control" acaba por se relacionar ao comando de Clone (git clone) do Git, da mesma forma que a opção de Commit faz referência ao comando git commit. Em complemento a tudo isso, a listagem a seguir mostra os demais comandos complementares (e suas equivalências no IDE) suportados atualmente pelo Delphi XE7.

- git checkout: comando que descarta toda e qualquer alteração realizada. No Delphi, se reflete na ação Revert, disponível no menu de contexto Git;
- git diff: comando que exibe as diferenças existentes entre as versões do conteúdo versionado. No Delphi, isso se reflete na guia Differences do History Manager;
- git annotate: comando que apresenta a idade relativa de cada linha presente no conteúdo do arquivo versionado, por meio de seu número de revisão. Tal opção fica disponível no History Manager do IDE, através do botão Annotate.

Assim sendo, em seu estágio atual, o Delphi acaba por contemplar os três modelos de versionamentos existentes neste contexto: "local" por meio do uso clássico do History Manager, "centralizado" através de sua integração com o Subversion e "distribuído" com o novo suporte a Git.

Mediante todos esses aspectos, o IDE comprova mais uma vez sua maturidade, uma vez que enfatiza diversos elementos extras, porém não menos importantes, que permeiam todo o cotidiano de desenvolvimento de software. Nesta linha se encontra o versionamento de código, que desde o Delphi XE, ganhou o enfoque necessário no âmbito da ferramenta, dada sua reconhecida importância. Por fim, assim como é corriqueiro dizer, um novo desafio está lançado, agora pela adoção do Git, nos cenários que lhe forem pertinentes.

Autor

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ e pós-graduado em Engenharia de Componentes. Atua como Analista Programador Delphi há 9 anos.

Links:

RAD Studio XE7 – Página oficial do produto
http://www.embarcadero.com/products/rad-studio

RAD Studio XE7 – Download Trial
https://downloads.embarcadero.com/free/rad_studio

Git – Download
http://git-scm.com/downloads

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

EMS: Conhecendo o Enterprise Mobility Services

Conheça esta nova solução de desenvolvimento do XE7

O Enterprise Mobility Services (EMS) é uma das grandes novidades trazidas pelo recente Delphi XE7 e representa um framework de desenvolvimento com capacidades REST (**BOX 1**), cuja finalidade principal é prover a devida mobilidade de serviços centralizados na nuvem. Em outros termos, o cenário estabelecido pelo EMS fundamenta um ambiente muito próximo ao fornecido por um servidor DataSnap REST. Assim, toda sua fundamentação se baseia num aspecto cliente/servidor, onde os serviços, dados e pontualidades de negócio se concentram no servidor, ficando devidamente expostos e disponíveis a clientes nativos Delphi e de outras tecnologias como Java, .NET, PHP, etc. A fim de pré-visualizar todo este novo cenário que se apresenta, a **Figura 1** mostra uma panorâmica geral de toda a solução proposta pelo contexto EMS.

BOX 1. REST

No cenário mais atual do desenvolvimento de software, REST é uma sigla bastante comum no meio Mobile e Web, e vem a resumir o termo em inglês Representational State Transfer que, numa tradução mais popular para o português seria algo como Transferência de Estado Representacional. Para fácil entendimento, REST pode ser definido como sendo um estilo de projetar aplicativos para internet, cuja estrutura deve se fundamentar em alguns aspectos relevantes, tais como: ser composta de elementos intercambiáveis e fracamente acoplados, prover recursos naturalmente nomeados e acessíveis diretamente através de uma URI (Identificador Uniforme de Recursos) própria.

EMS Server

Perante a composição do Enterprise Mobility Services, seu componente denominado EMS Server representa o lado servidor da arquitetura do framework, atuando como o canal direto de provimento das funções e serviços às aplicações clientes. Além disso, ainda neste contexto, outra atribuição importante atrelada ao Server diz respeito à sua atuação no gerenciamento do banco de dados nativo da solução, tratado como EMS Database, visto em detalhes mais adiante. A seguir são então elencadas de forma explícita as três funções essenciais do EMS Server neste cenário:

- Possibilita o registro de novos recursos, o que garante a extensibilidade do próprio servidor;
- Provê o acesso devido às informações do servidor a partir de uma aplicação cliente, por meio de uma API REST proprietária;
- Disponibiliza, também por meio de sua API REST, acesso às informações armazenadas na base de dados do EMS (EMS Database), tais como dados de usuário, grupos e dispositivos.

O servidor dispõe de uma API REST pré-construída, também tratada como Built-in API REST, que é devidamente exposta aos clientes, possibilitando o acesso a alguns de seus principais recursos e funcionalidades. Dentre essas, estão atividades relacionadas a dispositivos, usuários e toda a parte analítica elaborada e disponibilizada pelo próprio framework. Em complemento a esta pontualidade está a Custom REST API, que diz respeito aos novos recursos desenvolvidos para serem adicionados ao contexto do EMS Server existente. Aqui, tais recursos ganham o nome EMS Resources, sendo incorporados ao servidor por meio de pacotes específicos denominados EMS Packages.

Fique por Dentro

O presente artigo toma como foco o Enterprise Mobility Services, ou simplesmente EMS, que tem a finalidade de trazer a mobilidade de serviços em uma nuvem de forma simples aos projetos Delphi. Diante disso, aqui serão explicitados os principais conceitos técnicos relacionados ao tema, passando pelos detalhes de sua arquitetura, seus propósitos e, principalmente, as nuances que envolvem sua utilização na prática. A partir disso, o desenvolvedor estará apto a tomar decisões sobre a eventual adoção do framework em seu cotidiano construtivo e, eventualmente, sua implementação em ambientes reais de produção. Ainda seguindo por esta linha, este artigo servirá como fundação e ponto de partida para outros desenvolvimentos futuros sobre o assunto, dada a abertura e complexidade do tema.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

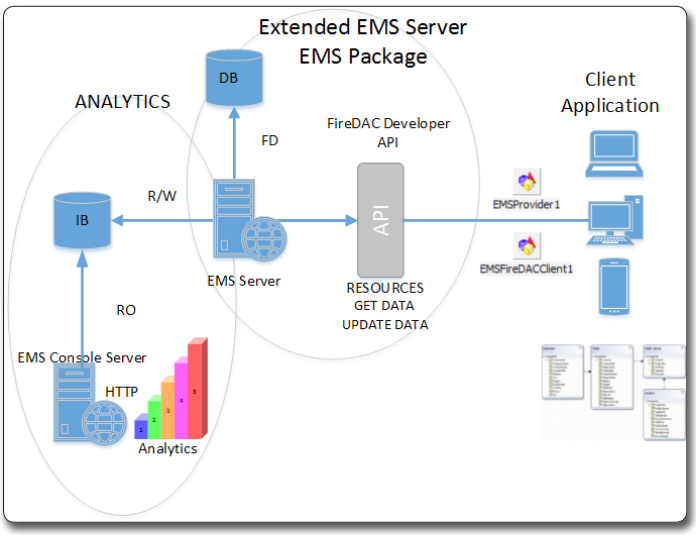

Figura 1. Enterprise Mobility Services – Visão geral do contexto

EMS Database

De forma padrão, a arquitetura do EMS prevê a presença invariável de um banco de dados nativo da solução, aqui tratado como EMS Database. Logo, é nele onde estarão armazenados os principais dados condizentes ao contexto, essenciais ao funcionamento linear do framework, tais como as informações de usuários e grupos de usuários de acesso ao servidor. Nesta etapa inicial de sua promissora trajetória, o Enterprise Mobility Services acaba por estabelecer o InterBase como sendo o principal e único SGBD adequado a essas pretensões. Isso se justifica pelo fato de ambos os produtos, EMS e InterBase, serem provenientes da mesma fabricante, a Embarcadero, o que dá margens para uma melhor integração neste seu momento primário de utilização. Obviamente, num futuro próximo, o leque de opções neste quesito deve ser expandido aos principais bancos de dados disponíveis no mercado. Por enquanto, na prática, ao tentar se configurar um ambiente EMS sem a presença de uma instância do InterBase instalada e operante, uma mensagem de erro é exibida, tal como mostra a **Figura 2**. Através de seus dizeres, fica nítida também a presença e utilização do FireDAC na constituição interna do framework.

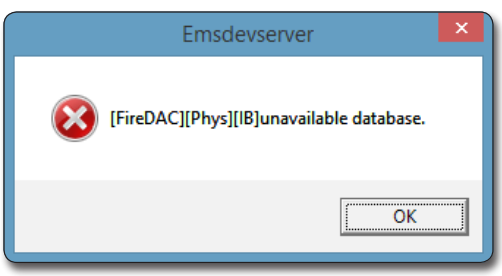

Figura 2. Mensagem de erro EMS – Servidor InterBase indisponível

Do ponto de vista deste cenário atual que é estabelecido, a barreira imposta pelo uso de um SGBD pago pode ser plenamente derrubada pela aquisição de sua versão gratuita. Disponibilizada como Developer Edition, a distribuição free do InterBase é um tanto quanto desconhecida por grande parte dos desenvolvedores da comunidade, muito em razão de seu ofuscamento pela massiva notoriedade do Firebird, enquanto SGBD gratuito. Um contraponto muito presente nesta questão se dá pela confusão em torno da versão free do InterBase com sua versão trial, uma vez que ambas podem ser obtidas de forma gratuita através do site oficial do produto (ver seção **Links** ao final do artigo). É importante então ressaltar que Trial caracteriza a distribuição de avaliação da versão mais completa do SGBD, denominada Server Edition, que contempla todos os recursos do produto, cujo uso é válido por tempo determinado. Numa linha oposta, a versão free caracteriza efetivamente uma versão singular do produto, de uso irrestrito, provida de um conjunto limitado de recursos e outras imposições. A principal delas se dá por sua incapacidade de deploy, cujo processo exige necessariamente a presença de uma das versões pagas do produto para seu uso em ambiente real de produção.

Da mesma forma que o IDE do Delphi, o release mais atual do InterBase Developer Edition até a escrita deste artigo é o XE7, cuja numeração equivale a 12.0. Neste mais novo lançamento do produto, algumas de suas limitações presentes até então foram retiradas, o que tende a contribuir de forma bastante positiva em sua eventual adoção em cenários locais e de estudo. Diante disso, a seguir são explicitadas algumas de suas características atuais:

- Suporte a conexões remotas via TCP/IP;
- Suporte a conexões remotas seguras via SSL;
- Suporte a criptografia em arquivos de backup do banco de dados;
- Trabalho com até 20 usuários simultâneos;
- Suporte até oito CPU Cores;
- Suporte de até quatro conexões por usuários;
- Número ilimitado de transações por conexão;
- Plataformas suportadas: Windows, Linux, Mac OS X e Solaris;
- Drivers de conectividade suportados: FireDAC, dbExpress, IBX, ODBC, JDBC, ADO.NET, PHP e Ruby.

EMS Environment

Na prática, o início da utilização do EMS é marcado pela configuração de seu ambiente, tratado neste contexto como EMS Environment. Para tal, são providos alguns recursos e aplicativos nativos distribuídos em meio à própria instalação do IDE, que acabam por facilitar todo o processo. Tomando como base uma instalação padrão em um sistema operacional Windows, a base de tais recursos é toda disponibilizada em uma pasta de nome "EMS", localizada em:

C:\Program Files (x86)\Embarcadero\Studio\15.0\ObjRepos\en\

Nela, são encontradas basicamente três estruturas, mais especificamente um diretório e dois arquivos, conforme pode ser visto na **Figura 3**. Ainda vale ressaltar que os recursos mencionados neste ponto fazem menção a estruturas físicas de recurso, tais

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

como diretórios e arquivos, não estando relacionados, portanto, aos EMS Resources.

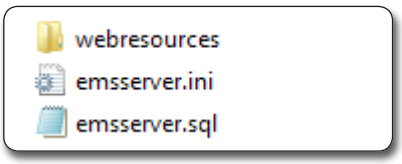

Figura 3. EMS – estrutura de recursos base

Para fins didáticos, diante deste cenário que se apresenta, a seguir são discorridos os detalhes técnicos relacionados a cada estrutura:

• webresources: pasta que contém um agrupamento de arquivos e scripts modelo, relacionados ao provimento do aplicativo web nativo do contexto. Dentre os elementos disponibilizados estão os scripts tidos como "padrão", a serem utilizados na renderização do aplicativo web, o qual é tido como um console web browser, uma vez que sua acessibilidade se dá por meio de um navegador de internet. Adicionalmente, a pasta webresources conta ainda com uma gama de arquivos adicionais, tais como os de imagem (.png e .ico), javascript (.js) e de estilo (.css), distribuídos em seus diversos sub-diretórios, e que acabam por fundamentar toda a parte relacionada ao layout do console Web. Logo, a modificação destes arquivos é que possibilitará ao desenvolvedor personalizar a aparência da aplicação;

• emsserver.ini: conforme sua extensão pressupõe, o arquivo emsserver.ini refere-se ao modelo de arquivo de configuração de um servidor EMS. Como exemplo, é nele onde estarão definidos alguns dos parâmetros de conexão do banco de dados. Adiante, na **Listagem 1**, que mostra um pequeno trecho do conteúdo do arquivo, é possível notar uma menção explícita ao InterBase;

• emsserver.sql: arquivo de script SQL contendo as instruções padrão para a criação do EMS Database. Novamente como exemplificação, na **Listagem 2** é mostrado um pequeno trecho do conteúdo deste arquivo, condizente ao script SQL de criação da tabela de usuários do contexto.

Além desses arquivos de apoio, que servirão como modelo para toda e qualquer nova construção EMS a ser realizada no ambiente de desenvolvimento corrente, são disponibilizados ainda dois aplicativos stand-alone, com finalidades distintas de suporte. EMSDevServer é o primeiro deles, e se incumbe de executar o servidor (EMS Server) propriamente dito, enquanto que EMSDevConsole possibilita a execução direta do EMS Console Server. Por estar relacionado à parte Server do âmbito do framework, a instalação do Delphi XE7 provê uma versão 32 e outra 64 bits, para cada aplicativo. Por padrão, a versão 32 é disponibilizada em:

C:\Program Files (x86)\Embarcadero\Studio\15.0\bin

EMS Development Server

O desenvolvimento de uma solução EMS acaba por envolver dois cenários um tanto quanto distintos. O primeiro, brevemente já mencionado, envolve o uso dos dois aplicativos stand-alone de suporte que acompanham a instalação do IDE, representativos do servidor e console EMS. Da mesma forma, uma abordagem alternativa se faz pelo uso de dlls similarmente representativas, que deverão ser alocadas em um servidor Web IIS (Microsoft Internet Information Services), que condiz com a única tecnologia suportada atualmente pelo Enterprise Mobility Services. Logo, seu envolvimento também se torna essencial num ambiente real de produção, já fazendo uma alusão ao processo de deploy, que envolve a alocação de toda a estrutura do EMS em um contexto real.

Listagem 1. Trecho do conteúdo modelo do arquivo emsserver.ini

```
[Data]
;# Interbase connection parameters
Database=[!DBPATH]
UserName=[!DBUSERNAME]
Password=[!DBPASSWORD]
```

Listagem 2. Trecho do conteúdo do script padrão do arquivo emsserver.sql

```
create table "USERS"
(
  "UID" INTEGER NOT NULL,
  "USERID" CHAR(36) NOT NULL UNIQUE,
  "USERNAME" VARCHAR(32) NOT NULL UNIQUE,
  "HPASSWORD" CHAR(32),
  "SALT" CHAR(6),
  "SESSIONID" CHAR(32),
  "SESSIONTIME" TIMESTAMP,
  "ISACTIVE"  BOOLEAN DEFAULT TRUE,
  "CREATED"  TIMESTAMP,
  "LASTMODIFIED" TIMESTAMP,
  "CREATOR" CHAR(36),
  PRIMARY KEY ("UID")
);
```

Ainda sobre o cenário de desenvolvimento, a **Figura 4** mostra o aplicativo EMS Development Server (EMSDevServer.exe) em ação. Pela própria denominação de sua janela, indicada como EMS Development Server, sua representatividade se dá por uma aplicação visual administrativa do EMS Server. Logo, através dela é possível realizar operações essenciais como:

• Iniciar o servidor EMS, por meio do botão Start;

• Parar o servidor EMS, por meio do botão Stop;

• Abrir o EMS Server no browser Web, por meio do botão Open Browser;

• Abrir o console do servidor (EMS Development Console), por meio do botão Open Console;

• Definir a porta de acesso ao servidor, via http, por meio da caixa de entrada Port;

• Habilitar/Desabilitar o log de mensagens do servidor, por meio da caixa de seleção "Enable logging";

• Limpar as mensagens registradas do servidor, por meio do botão Clear.

Além disso, o painel de Log presente na janela exerce papel fundamental e informativo às diversas pretensões gerenciais do

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

usuário/desenvolvedor perante o servidor, uma vez que nele é que ficará registrada toda e qualquer mensagem do EMS Server. Assim, seja uma simples informação de recurso registrado ou mesmo o indicativo de uma requisição (request) feita a partir de uma aplicação cliente, estarão presentes neste painel.

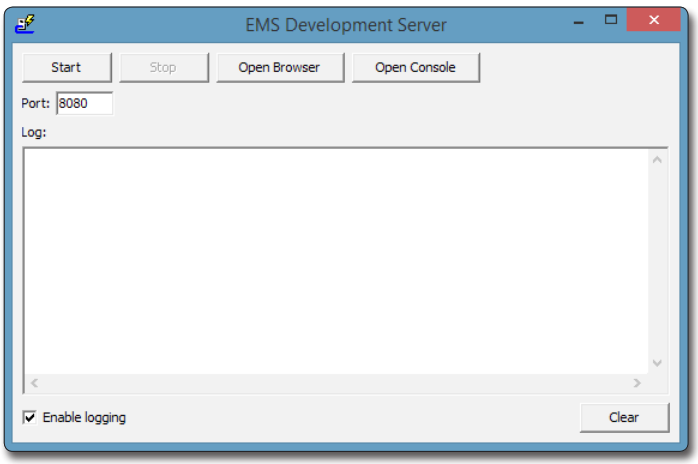

Figura 4. EMS Development Server

EMS Console Server

EMS Console Server é o nome dado a um serviço que dispõe toda a parte analítica do EMS Server e sua exibição natural ocorre em um Web Browser. Para atingir este fim, o console é fundamentado em três características:

• Acesso direto (somente leitura) ao banco de dados EMS para a obtenção das informações para a composição de análises;

• Composição de estatística panorâmica entre acesso à API, requisições, usuários e grupos de usuário;

• Disposição analítica de dados de resources de um modo geral, incluindo novos recursos, desenvolvidos para aumentar substancialmente a funcionalidade do servidor.

Uma vez acionado, o arquivo executável EMSDevConsole.exe dá origem à janela do console de desenvolvimento do servidor, oficialmente tratada como EMS Development Console (**Figura 5**). Esta então traz à tona algumas opções relacionadas ao EMS Server, uma vez que sua finalidade essencial é prover um canal de interação com o utilizador, de tal forma que ele obtenha de forma fácil informações administrativas como estatísticas de dados (EMS Database), informações de usuário, além de acesso aos recursos do próprio servidor. Assim, sem deixar de citar, suas opções de Start, Stop, Port e Open Browser acabam exercer a mesma função dos botões homônimos mostrados anteriormente na estrutura do EMS Development Server.

Configuração do ambiente EMS

A ação inicial relacionada ao uso do framework se dá em torno da configuração do ambiente EMS (EMS Environment), que irá estabelecer os pilares essenciais de todo seu ecossistema operacional. Mantendo a tradição que consagrou o Delphi, todo este processo é realizado de maneira visual, que o torna bastante intuitivo. Para tal, é disponibilizado ao desenvolvedor um Wizard, denominado EMS Setup Wizard, que o guiará por diversos passos instrutivos que darão forma à configuração desejada do ambiente. Na prática, a chamada deste assistente ocorre por meio do acionamento de qualquer um dos aplicativos stand-alone de apoio mencionados anteriormente. Nesta situação, a simples tentativa de execução do console, por meio do EMSDevConsole, ou do servidor propriamente dito, através do EMSDevServer, faz com que uma verificação seja feita no ambiente da máquina (sistema operacional), em busca de uma configuração válida ao ambiente EMS. Nos bastidores, tal verificação é realizada ainda no registro do Windows, isto porque, a cada nova configuração do EMS Environment realizada, uma chave de registro de sistema relacionada é criada. Por conseguinte, uma vez que nenhuma chave seja localizada, uma mensagem de diálogo é exibida (**Figura 6**), sugerindo a inicialização imediata do Wizard.

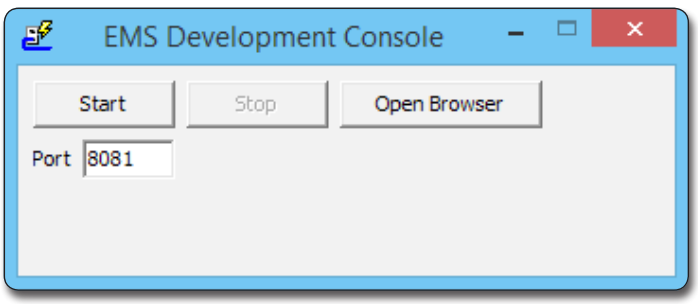

Figura 5. EMS Console Server

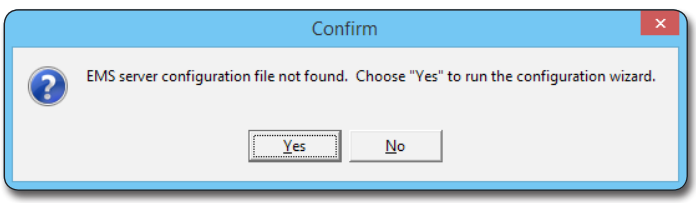

Figura 6. Mensagem de confirmação para inicialização EMS Setup Wizard

O assistente de configuração EMS

Com o intuito de colaborar no entendimento, a **Figura 7** exibe então o layout do passo inicial do assistente de configuração EMS. Nele estão fundamentadas as especificações relacionadas à criação do banco de dados do contexto, dado o alto grau de importância deste elemento no cenário.

A seguir são explanadas as finalidades de cada campo informativo apresentado:

•DB File Name: refere-se ao nome do arquivo de banco de dados. Novamente explicitando a presença do InterBase no processo, o próprio nome sugerido ao arquivo de banco de dados a ser criado apresenta a extensão ".ib", exclusiva deste SGBD;

•DB File Directory: indica não só o diretório para a alocação do arquivo de banco de dados, como também irá resultar no diretório padrão onde será criada toda a estrutura necessária ao funcionamento padrão do EMS. De forma mais específica, este diretório irá

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

conter todo o emaranhado base de arquivos e pastas, mostrados anteriormente na **Figura 3**;
•DB User Name: nome do usuário de acesso do EMS Database (padrão: SYSDBA);
•DB Password: senha de acesso do EMS Database (padrão: masterkey).

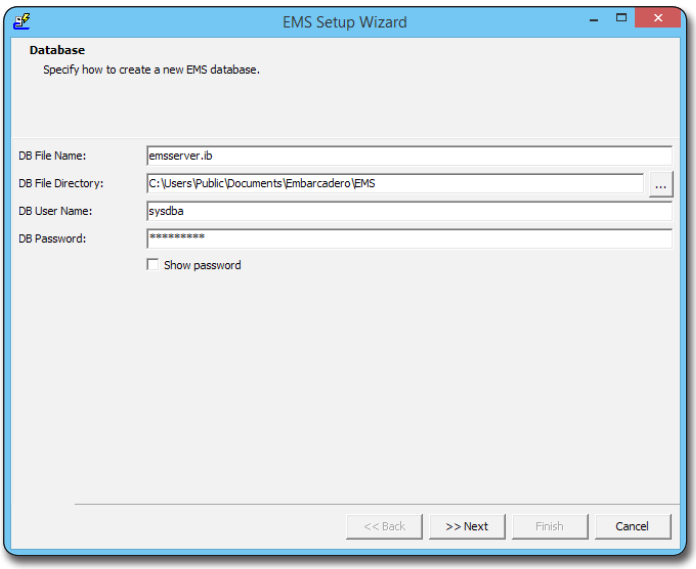

Figura 7. Passo inicial do EMS Setup Wizard

Com as configurações da base de dados definidas, o Wizard provê, em seu próximo passo, denominado Sample Data, a possibilidade da geração inicial e automática de alguns dados de amostra de usuários e grupos de usuário (**Figura 8**). Isso contribui de forma positiva no desenvolvimento inicial e testes de acesso ao servidor.

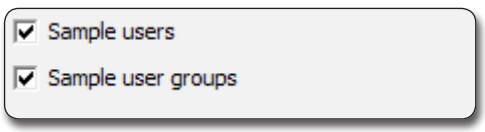

Figura 8. Opções para geração de dados de amostra para usuários e grupos de usuário

Este Wizard do EMS é composto essencialmente de quatro etapas simples. Assim, em complemento às duas primeiras, referentes à parte de dados do contexto, seu terceiro passo envolve a definição das credenciais de acesso ao EMS Console. Assim, conforme pode ser visto na **Figura 9**, duas são as informações requeridas neste ponto: Console User Name e Console Password. Tal como pode se supor, em razão de suas próprias nomeações, esses indicativos se referem às informações de usuário e senha, e trazem como padrão os valores "consoleuser" e "consolepass". Qualquer mudança aqui será persistida no arquivo de configuração do ambiente, que poderão ser comprovados mais adiante, ao término do processo.

Por fim, a quarta e última parte do assistente marca o ponto anterior à efetiva configuração de todo o ambiente EMS, sendo assim, algumas confirmações são exigidas, disponibilizadas novamente na forma de campos editáveis (**Figura 10**): DB File, Configuration File e Registry key. Sem deixar de mencionar, DB File refere-se ao caminho onde será criado o arquivo físico de banco de dados, definido ainda no primeiro passo do assistente. De forma semelhante, Configuration File indica o local de criação do arquivo de configuração (.ini) que irá conter toda a definição imposta ao ambiente EMS. Logo, em situações futuras, é por meio dele que ajustes pontuais poderão ser facilmente realizados nas diversas configurações do ambiente. Já o último campo, Registry key, está relacionado à citação anterior sobre a chave de registro que é criada a cada nova configuração do EMS Environment. Em vista disso, HKEY_CURRENTUSER\Software\Embarcadero\EMS\ConfigFile indica o caminho padrão sugerido para a criação da referida chave.

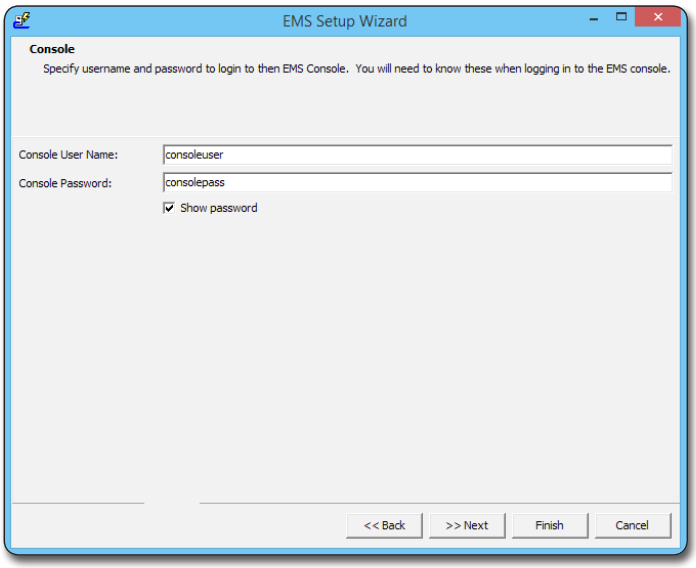

Figura 9. EMS Setup Wizard – 3ª etapa – credenciais de acesso ao Console

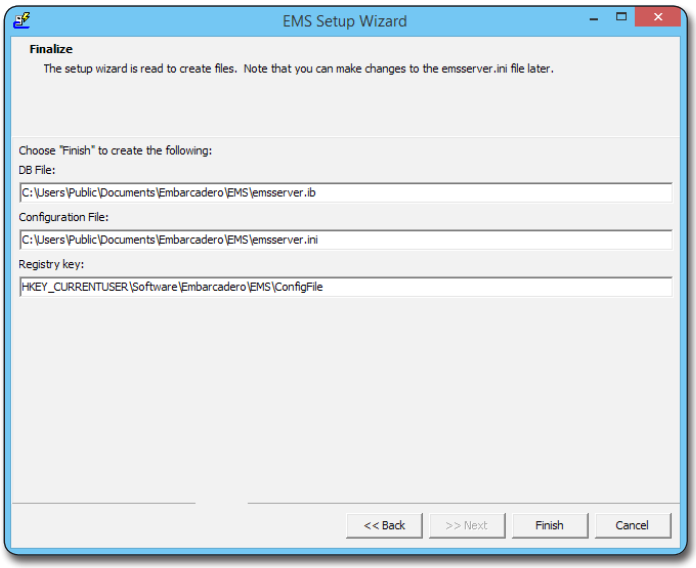

Figura 10. Finalização do EMS Setup Wizard

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

EMS Client

Diante do panorama imposto pelo EMS, o complemento ao funcionamento de seu servidor está relacionado às diversas e eventuais aplicações clientes que possam vir a integrar com este ecossistema, consumindo os serviços disponibilizados. Neste cenário, cada cliente é então tratado como um EMS Client Application, ou simplesmente EMS Client, e que tem como característica principal fazer uso de REST para consumir os serviços provenientes do EMS Server. Tal consumo acaba por incluir também a manipulação das informações oriundas de banco de dados, em processos habituais CRUD (Create-Retrieve-Update-Delete), que fundamentam a criação, recuperação, atualização e exclusão de dados. Nativamente, nos termos do XE7, um cliente EMS nada mais é do que uma simples aplicação Desktop ou Multi-Device, provida de um componente específico.

TEMSProvider

Ainda sob o contexto da nova versão do Delphi, o componente tido como peça-chave para a construção de aplicações EMS cliente surge também como uma das grandes novidades deste release. Sob a denominação TEMSProvider, ele fica disponível no grupo BAAS Client da Tool Palette (**Figura 11**), e tem a companhia de outro elemento exclusivo ao EMS, denominado TEMSFireDACClient, o qual será tratado mais adiante. De forma simples, a função essencial do TEMSProvider é configurar a conexão com o EMS Server, usando basicamente para isso, três de suas propriedades:

- URL Host: indica a URL do servidor a ser conectado. Neste ponto, essencialmente dois são os valores a serem utilizados – "localhost" para ambientes locais (EMS Server rodando na mesma máquina do cliente) ou o endereço IP, para ambientes remotos (servidor e cliente rodando em máquinas diferentes);
- URL Port: informa a porta de conexão com o servidor;
- URL Protocol: diz respeito ao protocolo a ser usado nas solicitações para o servidor. HTTP e HTTPS são as opções pré-disponibilizadas.

Em complemento, a **Figura 12** mostra uma configuração de exemplo destas propriedades, tomando como base o desenvolvimento inicial num ambiente local.

Testando a conectividade de um cliente nativo EMS

Olhando pelo lado prático, diante desta configuração linear de uma aplicação cliente nativa EMS, o passo seguinte a esta ação se dá pelo simples teste de conectividade com o EMS Server, cujo o resultado irá determinar a plena continuidade de todo o desenvolvimento. Em vista disso, uma forma bastante simples e eficaz para a realização desta atividade se dá por meio do próprio componente responsável pela conexão em questão, o TEMSProvider. Assim, tomando como base um projeto cliente do tipo Multi-Device Application, que a partir desta versão reflete a nova denominação dada a projetos FireMonkey (FMX), uma opção específica disponibilizada no menu de contexto do componente provê este tipo de teste. Conforme pode ser visto na **Figura 13**, tal opção apresenta-se sob o nome de "Test Connection", acessível ainda em tempo de design. Uma vez acionada, o sucesso desta operação resulta então numa resposta do servidor na forma de uma mensagem de diálogo, tal qual mostrada na **Figura 14**. Do ponto de vista técnico, a opção Test Connection do componente efetivamente realiza uma invocação a um serviço padrão do servidor EMS, cuja implementação se dá de forma nativa pelo seu próprio framework.

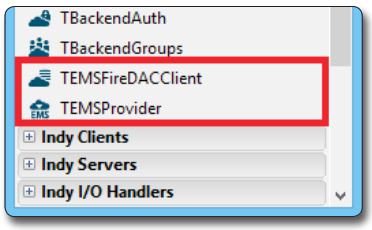

Figura 11. Componentes para EMS

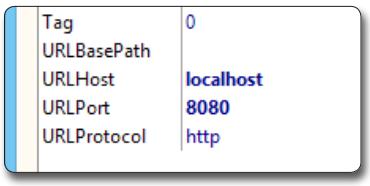

Figura 12. TEMSProvider – configuração de suas propriedades

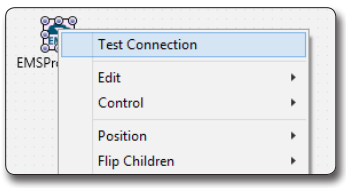

Figura 13. TEMSProvider – opção Test Connection

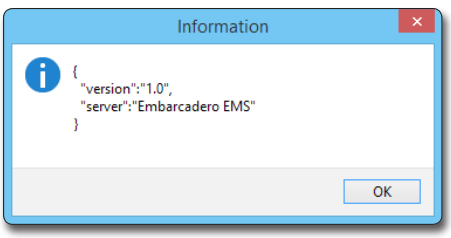

Figura 14. Teste de conectividade cliente – mensagem de sucesso

Testando a conectividade ao EMS Server via browser

Adicionalmente, a fim de justificar a característica REST de acessibilidade do EMS Server, o mesmo serviço utilizado pelo TEMSProvider como base para o teste de conectividade ao servidor pode ainda ser invocado de inúmeras outras formas, incluindo a mais tradicional, via browser. Sendo assim, a seguir na **Figura 15**, é mostrada a mesma invocação feita anteriormente, porém sob outro aspecto de acesso, agora via URL explícita. Em vista disso, o retorno agora se dá no formato JSON (**BOX 2**), já habitual na comunidade dada a crescente adoção de servidores DataSnap REST. Por fim, os detalhes a serem observados aqui ficam por conta da denominação do referido serviço (nomeado "version"), bem como as informações de Host e Porta utilizadas

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

anteriormente na configuração do componente do Delphi (localhost e 8080, respectivamente), que juntos acabam por compor a URL utilizada – localhost:8080/version.

Sem deixar de mencionar, este tipo de teste extrapola o cenário nativo de aplicações clientes Delphi, passando para uma perspectiva mais abrangente, envolvendo qualquer tipo de tecnologia com suporte a REST. Isso vem a confirmar o contexto de "mobilidade" proposto pela solução.

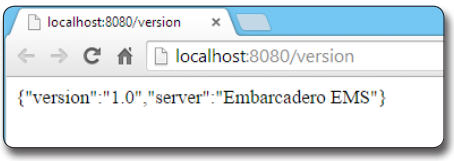

Figura 15. Acesso ao serviço "version", via browser

BOX 2. JSON

JavaScript Object Notation, ou simplesmente JSON, deve ser entendido como um formato bem definido para intercâmbio de dados para os mais variados fins. De característica leve e de fácil entendimento tanto para máquinas quanto para humanos, ele tem sua composição baseada em um subconjunto da linguagem de programação JavaScript. Na prática, JSON é formatado em texto puro, independente de idioma, porém com uso de convenções bastante familiares e intuitivas ao âmbito dos programadores de software. No cenário Delphi, este formato de dados ganhou destaque natural com o surgimento dos servidores DataSnap REST, sendo então amplamente utilizado pela comunidade de desenvolvedores.

EMS Package e EMS Resource

No âmbito do Enterprise Mobility Services, a busca pela ideal conectividade entre cliente e servidor só é justificada pelo consumo, por parte das aplicações cliente, dos recursos necessários para sua própria dinâmica de negócio, disponibilizados de forma centralizada pelo EMS Server. Diante disso, é válido estabelecer neste ponto que a API padrão provida pela arquitetura do servidor EMS e que fica exposta nativamente aos clientes desde o primeiro momento de sua construção, acaba por se limitar a disponibilizar serviços relacionados apenas à parte administrativa, envolvendo análises estatísticas e de dados do EMS Database. Logo, mediante este cenário, fica implícita a necessidade pela extensão dos recursos providos pelo servidor, a fim de atender os mais variados nichos de negócio a que a solução se propõe a atender. Em outras palavras, a partir deste ponto, é chegada a hora da implementação de seus próprios recursos. Em vista disso, nota-se o aparecimento de dois novos conceitos: EMS Package e EMS Resource.

EMS Package

Nesta conjuntura um pacote EMS, ou simplesmente EMS Package, pode ser definido como sendo o elemento que proporciona a extensibilidade de um servidor EMS. Para isso, cada novo recurso (EMS Resource) definido deve estar essencialmente registrado em um pacote deste tipo que, ao ser integrado a um EMS Server, irá então fundamentar a extensão de suas funcionalidades. Dada sua importância, o próprio IDE do Delphi provê uma nova opção exclusiva para a criação de um novo pacote, que fica disponível em seu repositório de projetos (File > New > Other > Delphi Projects > EMS > EMS Package).

Seguindo a tradição do Delphi, o acionamento desta opção dá origem a um breve assistente, denominado EMS Package Wizard, que irá guiar o desenvolvedor, passo-a-passo, na construção da estrutura de um EMS Package. Novamente, em vista da relevância deste elemento para o contexto, logo em sua etapa inicial (**Figura 16**) uma importante decisão deve ser tomada com relação ao tipo de pacote a ser criado.

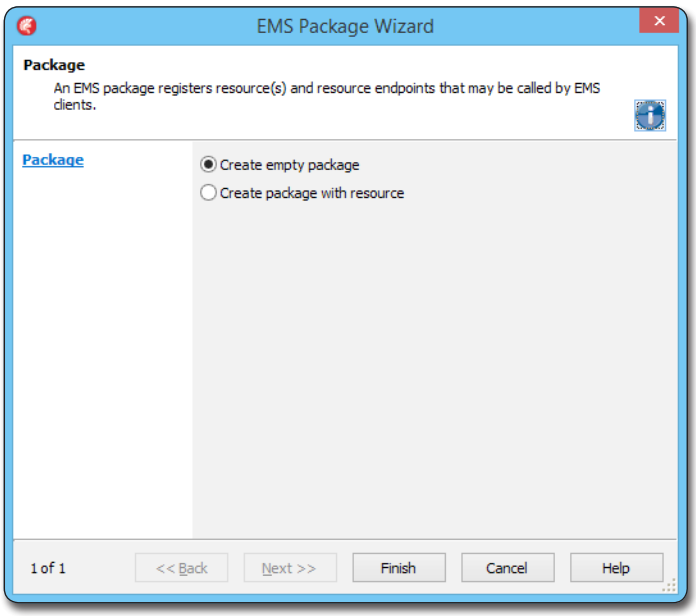

Figura 16. Etapa inicial do EMS Package Wizard

Para isso, a escolha é determinada por duas opções bastante distintas:

- Create empty package: conforme sua indicação, esta opção cria um pacote vazio, o que implica dizer que nenhum recurso será registrado no servidor num primeiro momento. Esta ação se torna útil mediante a experiência adquirida pelo uso do EMS, o que garante uma maior liberdade ao desenvolvedor, em adicionar eventuais recursos ao pacote a qualquer momento do desenvolvimento;
- Create package with resource: ao contrário da primeira opção, esta efetivamente cria um pacote contendo um recurso. Tendo em vista este fundamento, a **Figura 17** mostra os parâmetros providos no passo posterior do assistente, onde deverá ser informado o nome do recurso a ser criado, bem como o tipo do arquivo, se Unit, para o uso de código puro, ou Data Module, para a utilização de componentes visuais.

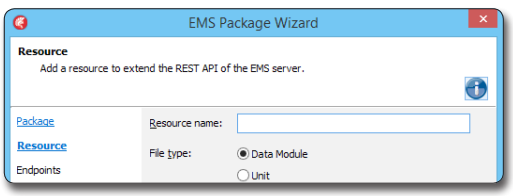

Figura 17. EMS Package Wizard – Configuração do recurso a ser criado

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Em complemento à opção "Create package with resource", a última parte do Wizard promove a escolha dos métodos Endpoints que serão adicionados ao recurso. Neste quesito, oito são as opções: Get, GetItem, Post, PutItem e DeleteItem. Tendo em vista a abrangência de mais este conceito relacionado, a seguir é feita uma breve explanação sobre o assunto.

Ainda é válido ressaltar que no cenário Delphi, o conceito de pacote (ou package) já é bastante tradicional, dado o tipo de projeto Package, que resulta num arquivo ".bpl", muito utilizado na modularização de aplicações e construções de componentes. Logo, é importante evidenciar que um EMS Package nada mais é do que um pacote tradicional, porém direcionado às suas pretensões. Isso pode ser comprovado no projeto resultante dos passos do assistente, que dispõe da estrutura tradicional de um projeto Package, incluindo sua extensão ".bpl" e os sub-diretórios Requires e Contains. (**Figura 18**).

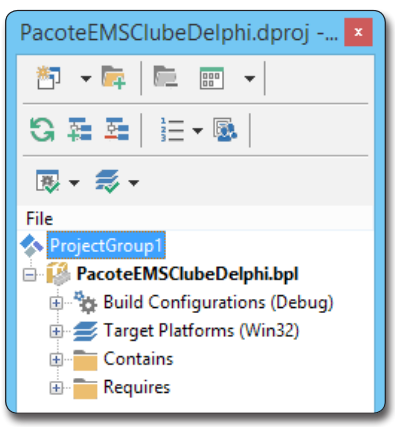

Figura 18. EMS Package – estrutura do projeto

EMS Resource Endpoints

Os citados métodos Endpoint (Endpoint methods) fazem referência então a outro conceito envolvido neste cenário, denominado EMS Resource Endpoints. Por definição, cada resource deste tipo equivale a um REST endpoint, que simplesmente mapeia de forma direta um determinado método HTTP. Mediante a amplitude REST, quatro são os métodos que compõem o conjunto de HTTP methods permitidos e usuais – GET, PUT, POST e DELETE – alusivos às operações CRUD (Create-Retrieve-Update-Delete). Sendo assim, cada EMS Resource definido em um pacote EMS poderá expor os seguintes métodos endpoints:

- Get: recupera dados a partir de um recurso tendo GET como o tipo de solicitação HTTP associado;
- GetItem: recupera os dados de um item específico. Também tem o GET como o tipo de solicitação HTTP associado;
- Post: atualiza dados a partir de um recurso, tendo o POST como o tipo de solicitação HTTP associado;
- PutItem: atualiza um item de um recurso usando uma solicitação HTTP PUT;
- DeleteItem: deleta um item de um recurso usando uma solicitação HTTP DELETE.

EMS Resource

O início da implementação de um novo EMS Resource está associado à criação de um EMS Package. Além disso, na prática, a criação de um novo recurso acaba por determinar a definição de uma nova classe de codificação. A fim de dar mostras para tal afirmação, a seguir, na **Listagem 3**, é mostrado o trecho de código da declaração de uma nova classe denominada TOlaClubeDelphiResource, referente a um recurso nomeado como OlaClubeDelphi, criado com todas as opções de métodos Endpoints habilitadas.

Listagem 3. Declaração da classe de resource – TOlaClubeDelphiResource

```
type
 [ResourceName('olaclubedelphi')]
 {$METHODINFO ON}
 TOlaClubeDelphiResource = class
 published
  procedure Get(const AContext: TEndpointContext;
  const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
  [ResourceSuffix('{item}')]
  procedure GetItem(const AContext: TEndpointContext;
  const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
  procedure Post(const AContext: TEndpointContext;
  const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
  [ResourceSuffix('{item}')]
  procedure PutItem(const AContext: TEndpointContext;
  const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
  [ResourceSuffix('{item}')]
  procedure DeleteItem(const AContext: TEndpointContext;
  const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
 end;
 {$METHODINFO OFF}
```

Tendo em vista o trecho de código mostrado, a fim de facilitar a identificação do recurso e seus métodos endpoint, o EMS adota três marcações distintas, tidas como padrão, para todo código criado:

- [ResourceName('nome')]: marcação usada para identificar o nome do recurso, logo, este é o termo que será utilizado para sua invocação por meio de uma chamada via URL. Para este exemplo, diante da marcação utilizada [ResourceName('olaclubecelphi')], sua URL de chamada resultaria em algo como http://localhost:8080/olaclubedelphi, tendo em vista uma abordagem local;
- [ResourceSuffix('{item}')]: marcação que representa o parâmetro utilizado no método endpoint. Numa chamada essencial, o valor do parâmetro é posicionado na extremidade da URL correlata, tal como <host>:<porta>/resource/parâmetro;
- [EndpointName('<endpointname>')]: marcação que expressa a denominação do método endpoint que ficará exposta na interface do EMS Console.

FireDAC EMS Resource

Em complemento aos conceitos de EMS Package e EMS Resource, o FireDAC EMS Resource surge como um recurso EMS relacionado à API do FireDAC, o que provê o devido suporte a banco

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

de dados e as diversas manipulações envolvidas em decorrência disto. Além disso, neste sentido, o contexto proporcionado pelo Enterprise Mobility Services tende a envolver mais de uma base de dados, tal como o cenário esboçado ainda na **Figura 1**.

Conforme é ilustrado na imagem, há uma clara distinção entre o banco de dados de retaguarda do servidor e o banco de dados relacionado a recursos. O primeiro é o então denominado EMS Database, caracterizado como o banco de dados padrão de toda a arquitetura EMS. Conforme já citado, trata-se então de uma base essencialmente InterBase, cuja essência é armazenar os dados de usuários, grupos de usuários e outras informações de análise relacionadas. Em contraponto, um banco de dados relacionado a um recurso pode ser de qualquer tipo suportado pelo FireDAC, o que acaba por envolver os principais SGBDs do mercado, tais como o próprio InterBase, Firebird, Oracle, SQL Server, MySQL, PostgreSQL, entre outros. Além disso, sua função é adequada aos propósitos do negócio, o que fica a cargo do desenvolvedor, logo, sua usabilidade é a mesma que em um desenvolvimento de uma aplicação de banco de dados tradicional. Sem deixar de mencionar, a própria denominação adotada para este conceito – FireDAC EMS Resource – reforça ainda mais a definitiva adoção, por parte da Embarcadero, do FireDAC como sendo a principal opção de biblioteca de acesso a dados nativa atualmente no IDE.

Isto posto, já olhando pelo aspecto prático do assunto, a composição de um FireDAC EMS Resource acaba por envolver algumas particularidades, exigidas pelo próprio âmbito ao qual este tipo de recurso se insere. Tais nuances acabam por diferir sua implementação da construção de um EMS Resource simples. A fim de facilitar o entendimento, a seguir são expostos os principais detalhes relacionados.

Data Module

A criação de um novo resource envolve também a definição de um novo pacote, que no Delphi XE7 é feito por meio do assistente EMS Package Wizard. Durante este processo, uma das etapas exige o apontamento do tipo de arquivo que estará associado ao recurso criado, cujas opções são Unit, para uma abordagem envolvendo somente código, ou Data Module, quando há o envolvimento de componentes. Diante disso, para a criação de um FireDAC EMS Resource, é fundamental a escolha pela opção Data Module, uma vez que diversos serão os componentes FireDAC envolvidos, necessários à conectividade e manipulação do banco de dados.

Banco de dados e componentes

Outro fator de diferencial para a criação de um FireDAC EMS Resource se dá pela invariável presença de um banco de dados relacionado, que justifica a criação de um recurso deste tipo. Por este fato, três outros componentes surgem como essenciais nesta conjuntura. O primeiro e mais importante é o TFDConnection, responsável pela efetiva conexão com o banco. Sendo assim é nele onde estarão definidos os parâmetros de conectividade que serão utilizados, o que inclui informações como o caminho físico da base de dados, usuário e senha de acesso, bem como a identificação do driver relacionado. De forma ilustrativa, a seguir é mostrado na **Listagem 4** um breve exemplo de configuração da propriedade Params do referido componente, tomando como base o uso de um SGBD Firebird, e sua base de exemplo EMPLOYEE.FDB.

Adicionalmente o uso de um TFDConnection em um cenário de aplicação implica a adição de outros dois elementos no contexto: um relacionado ao driver utilizado e outro denominado TFDGUIxWaitCursor, cuja implementação é obrigatória para aplicações FireDAC. Por padrão, o componente de driver assume a nomenclatura TFDPhysXXDriverLink, onde XX refere-se ao ID do Driver em questão. Tomando o Firebird como exemplo, o componente associado é o TFDPhysFBDriverLink. De forma abrangente, os componentes deste tipo ficam disponíveis no grupo FireDAC Links da Tool Palette do IDE. Na prática, o uso de um DriverLink internamente à aplicação faz com que nenhuma biblioteca do FireDAC necessite ser distribuída junto ao cliente. O uso efetivo destes dois componentes físicos pode plenamente ser substituído pela simples declaração de suas units relacionadas na seção uses do projeto.

Todavia, no que tange os componentes, a grande particularidade aqui fica por conta do uso de outros dois elementos, TFDSchemaAdapter e TFDStanStorageJSONLink, cuja funcionalidade básica é atuar como um repositório central de dados armazenados em cache e lidar com toda a parte JSON envolvida, respectivamente. Por fim, a seguir, na **Listagem 5**, é exibida uma pequena implementação de exemplo de um método endpoint Get relacionado ao recurso. Para um FireDAC EMS Resource, tal método representa a obtenção de um conjunto de dados provindos do banco no formato JSON, que é o formato padrão para o contexto REST que o EMS visa atender.

Listagem 4. Configuração

```
Database=C:\Program Files (x86)\Firebird\Firebird_2_5\examples\empbuild\
EMPLOYEE.FDB
User_Name=SYSDBA
Password=masterkey
DriverID=FB
```

Listagem 5. FireDAC EMS Resource – método endpoint Get

```
procedure TTestedbResource1.Get(const AContext: TEndpointContext;
const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
var
  LocalStream: TMemoryStream;
begin
  LocalStream := TMemoryStream.Create;
  FDQuery1.Open;
  FDSchemaAdapter1.SaveToStream(LocalStream, TFDStorageFormat.sfJSON);
  AResponse.Body.SetStream(LocalStream, 'application/json', True);
end;
```

Nesta amostragem, o componente Adapter atua como o elemento responsável por adequar, para o formato JSON, os dados provindos do banco, que são então mantidos num elemento de Stream local. Por conseguinte, este é o elemento que será tomado como base para a construção da resposta do método, a qual será enviada aos clientes solicitantes.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

TEMSFireDACClient

No cenário EMS, diante de um resource FireDAC (EMS FireDAC Resource) definido no servidor, o componente TEMSFireDACClient surge como o elemento de destaque na construção de um cliente nativo. Isto porque ele é o responsável por se conectar a um TFDSchemaAdapter a fim de obter os dados que estarão envolvidos na interação cliente/servidor. Tal ligação se dá por meio de sua propriedade SchemaAdapter e é complementada por outras duas – Provider e Resource – que especificam o componente TEMSProvider envolvido, bem como o nome do resource que se deseja acessar.

Testando um recurso criado

O teste imediato de um novo recurso EMS criado, como era de se esperar, pode ser feito ainda no momento de seu desenvolvimento, graças ao auxílio provido pelo próprio IDE. Vale novamente ressaltar que todo recurso EMS está contido em um pacote, cuja extensão é ".bpl", o que inviabiliza sua direta execução. Sendo assim, para o seu processo de teste, uma aplicação hospedeira deve estar em execução que, neste caso, tende a ser a própria aplicação stand-alone EMS representativa do servidor (EMSDevServer.exe). Logo, esse tipo de configuração é feito por meio da opção Run > Parameters, no do IDE, contexto do próprio projeto EMS Package, conforme mostra a **Figura 19**. Na sequência do processo, a simples invocação da opção Run Without Debugging (Shift + Ctrl + F9) do IDE faz com que o pacote criado seja carregado junto da aplicação host, tornando o resource disponível, podendo o mesmo ser chamado via URL, tal como mostra a **Figura 20**. Nesta imagem é possível notar ainda o retorno no formato JSON, do método Get mostrado anteriormente, o qual poderá ser consumido por qualquer tecnologia com o devido suporte, o que representa a grande maioria das tecnologias de desenvolvimento atuais. Sem deixar de mencionar, os dados mostrados são equivalentes às informações da tabela COUNTRY do banco Employee do Firebird, obtido por um simples TFDQuery.

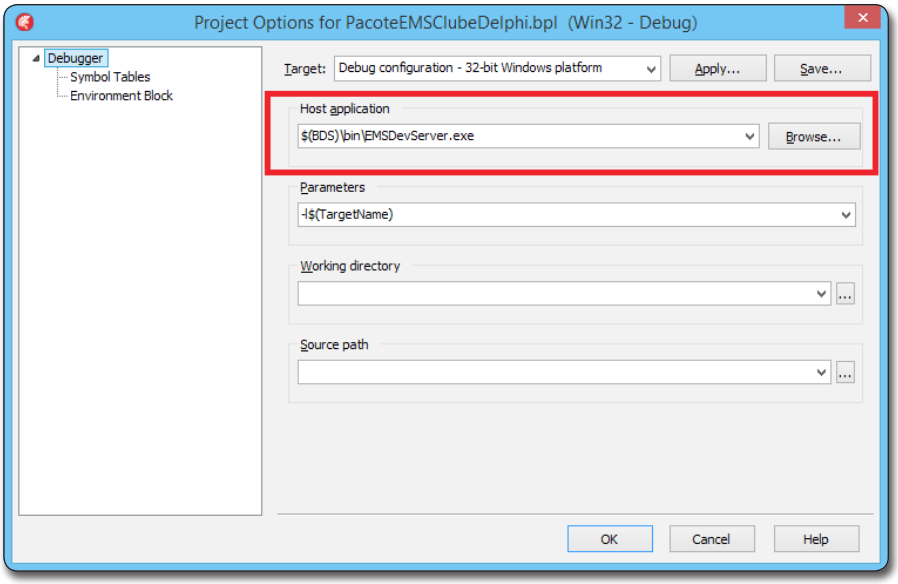

Figura 19. Configurando a aplicação hospedeira

Figura 20. Chamando o método Get do recurso "testedb"

Na versão XE7, apesar do chamariz para o desenvolvimento nativo multi-device, o real atrativo se dá pela estabilização definitiva de muitos dos recentes recursos adicionados, que no final das contas resulta numa maturidade abrangente por parte da própria ferramenta.

Em meio às novidades do XE7, um item bastante promissor e que acaba por passar quase que despercebido é o Enterprise Mobilty Services, ou simplesmente EMS, que surge como uma alternativa ao DataSnap, no que diz respeito à disponibilização de serviços centralizados a clientes heterogêneos. Tal como pode ser visto neste artigo, toda sua força motriz é baseada numa API REST bastante eficiente, que tende a ser customizada pelo próprio desenvolvedor, a fim de se atingir os mais variados objetivos. Ao mesmo tempo, a Embarcadero dá novamente mostras de sua preocupação em manter seu produto principal alinhado às mais recentes tendências de mercado e onde termo "mobilidade" se mostra cada vez mais presente.

Autor

Fabrício Hissao Kawata

fabricio.kawata@bol.com.br

Formado em Processamento de Dados pela FATEC-TQ e pós-graduado em Engenharia de Componentes. Atua como Analista Programador Delphi há 9 anos.

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Atualizando aplicações com TWebUpdate

Conheça os principais recursos deste poderoso componente da TMS Software

No âmbito comercial do cenário de software, a plena existência de um aplicativo rodando nas diversas estações de usuário, acaba por exigir uma série de fatores que por vezes ultrapassam as questões de negócio e mercadológicas. Um bom exemplo disso se dá pela própria atualização do software em questão, que acaba por envolver diversos aspectos que fogem do escopo central ao qual a aplicação visa atender. Logo, independente do aplicativo controlar Vendas, Estoque, Fluxo de Caixa ou servir como um utilitário de integração de dados ou envio de e-mails, é certo que uma inteligência concorrente para sua atualização, quando necessária, deverá existir.

Exatamente neste ponto é que surge então o TWebUpdate, um componente não-visual da empresa TMS Software, cujas capacidades são traduzidas nos diversos recursos disponibilizados para uso, estritamente relacionados à atualização de arquivos de modo geral. Logo, sua abrangência acaba por cobrir toda a área de software e seus arquivos relacionados, o que inclui o próprio executável de uma aplicação. Assim, na prática, sua utilização está muito ligada ao cenário de "atualizadores", servindo como elemento central na construção de utilitários desse tipo.

Fique por Dentro

É muito comum um mesmo sistema ser disponibilizado em vários clientes, ou, em um cliente que contenha várias máquinas. A tarefa de atualização dessas máquinas pode ser uma tarefa não muito fácil quando se trata de um aplicativo desktop, onde a execução ocorre localmente. Em um ambiente WEB a execução é centralizada no servidor onde o aplicativo está hospedado e sua atualização é simples, basta substituir a versão antiga pela nova no servidor, que todos clientes estarão atualizados. Mas e quanto ao cenário desktop? É nesse contexto que o componente TWebUpdate age, trazendo recursos que facilitam a criação de aplicações que se auto atualizam não só pela WEB, mas por outros meios como FTP e pastas compartilhadas em rede. Veja neste artigo como toda complexidade é abstraída, simplificando a atualização de aplicativos.

TMS Software

A TMS Software desenvolve componentes para os mais variados fins. Em tempos mais recentes, seu nome esteve cada vez mais envolvido no cenário Delphi, a partir de sua parceria com a Embarcadero, cujo resultado se deu na forma de cessão de pacotes de componentes aos proprietários de licenças da ferramenta.

Seu extenso leque de componentes é direcionado às duas principais plataformas de desenvolvimento da ferramenta – Visual Component Library (VCL) e FireMonkey (FMX) – atendendo a cenários Desktop, Mobile e Web. TDBAdvSpinEdit, TDBAdvNavigator, TAdvSmoothCalculator e TPlannerCalendar são alguns exemplos de componentes. Ainda relacionado ao Delphi, outro produto TMS já bastante popular em meio à comunidade é o TMS Aurelius, um framework ORM (mapeamento objeto-relacional), cuja abordagem pode ser encontrada em edições anteriores desta mesma revista.

TMS WebUpdate

O TMS WebUpdate é um componente essencialmente VCL, parte integrante do TMS Component Pack suportado pelo Delphi 7. Em razão do nome de sua classe, comercialmente, o componente é tratado como TWebUpdate. Devido à sua característica fundamental, sua aplicação prática é toda direcionada ao desenvolvimento de aplicações Desktop, Win32 quanto Win64.

Sua função básica é prover a distribuição de arquivos de forma rápida e facilitada, com base nas tratativas de transferência de arquivos já tradicionais no âmbito tecnológico: via rede, FTP ou HTTP/HTTPS. Adicionalmente, o formato de distribuição envolvido pode envolver quatro opções relevantes, sentenciadas por arquivos normais, arquivos de diferença (patch difference files), arquivos CAB ou arquivos compactados LZ.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Apesar de poder lidar com arquivos de um modo geral, toda a popularidade do TWebUpdate se volta à atualização de aplicações, deixando a plena impressão de sua exclusividade para este ponto. Isso se justifica dada sua excelência em lidar com transferência de arquivos e recorrência do tema em meio à comunidade de desenvolvedores, visto que esse tipo de "atualização" sistêmica se mostra como uma constante em ambientes corporativos e comerciais. Além disso, do ponto de vista comercial, toda essa caracterização imposta ao TWebUpdate, relacionada à exclusiva atualização de aplicação, faz com que ele se torne um dos grandes nomes, em termos de componente, atualmente disponíveis no mercado.

Principais características

Tomando como base as proposituras expostas pela própria fabricante, o TWebUpdate acaba por fundamentar um emaranhado de recursos e otimizações, que dão forma às suas principais características:

- Simples e de fácil utilização;
- Extensa capacidade de personalização para o processo de atualização de aplicativos;
- Plenamente utilizável para atualização de aplicativos, componentes e dados;
- Recurso de detecção automática de novas versões da aplicação;
- Mecanismo de auto atualização;
- Download de novas versões via rede, FTP ou HTTP/HTTPS;
- Trabalho com várias formas de detecção de novas versões, cuja verificação pode ser baseada em diversos aspectos como recursos, data, checksum e tamanho de arquivo;
- Suporte a formas customizadas de verificação de versão;
- Suporte a atualizações parciais;
- Mostra de informações "What's new" (o que há de novo) durante o processo de atualização;
- Tratativas de UAC do sistema operacional (Windows);
- Inclui assistente (Wizard) opcional interno que guiará o usuário através do processo de atualização da aplicação;
- Inclui utilitário de suporte (Update Builder) para criar e automatizar a geração de arquivos de controle de atualização;
- Produção de logs e estatísticas.

Download e instalação

O TWebUpdate, assim como a grande maioria dos componentes produzidos pela TMS, é um produto pago, que acaba por envolver uma licença comercial de uso. Neste ponto, o componente pode ser adquirido tanto de forma individual, quanto via TMS Component Pack, sendo parte integrante do pacote que engloba os principais componentes TMS. Tal qual já é de praxe em produtos comerciais, uma versão gratuita de avaliação do TWebUpdate é disponibilizada ao grande público (ver seção **Links** ao final do artigo), sendo esta a opção ideal para os contatos iniciais com o componente.

Diferente de alguns componentes, a instalação do TWebUpdate ocorre de forma bastante simplificada, por meio do acionamento de um simples executável, que dá origem a um Wizard de instalação já tradicional no meio tecnológico. Tal assistente guiará o usuário pelos passos básicos relacionados à instalação do produto, se encarregando em realizar todo o processo necessário para a adequação do mesmo ao IDE corrente, deixando tudo transparente ao desenvolvedor. Assim, sem deixar de citar, a **Figura 1** mostra a tela inicial do TMS WebUpdate Setup Wizard, num ambiente contendo uma instalação do Delphi XE7 (RAD Studio XE7).

Figura 1. Tela inicial do Wizard de instalação do TMS WebUpdate for RAD Studio XE7

Uma vez instalado, o TMS WebUpdate dá origem a um novo grupo de componentes na Tool Palette, denominado TMS Web (**Figura 2**). Nele ficam então concentrados diversos elementos que acabam por se inter-relacionar com o principal – o TWebUpdate. Assim, em sua essência, a efetiva utilização de cada um dos elementos sufixados com "UpdateWizard" estará condicionada a presença de um TWebUpdate operante.

Funcionamento básico

O processo de atualização de uma aplicação imposto pelo TWebUpdate acaba por abranger várias etapas sequenciais distintas, iniciada com a obtenção de um elemento informativo, tratado aqui como um arquivo de controle com a extensão .inf, que é obtido a partir de um local base. Na sequência o conteúdo do mesmo é processado, a fim de se verificar algumas informações relevantes ao contexto, relativas a arquivos, ações e versões. Por fim, há então a efetiva atualização da aplicação, que se resume na extração e cópia do novo executável e seus eventuais arquivos complementares (Ex. Bpls, DLLs, etc.). Nesta fase, de forma opcional, o próprio componente provê recursos para que a aplicação seja automaticamente encerrada, a fim de se evitar qualquer tipo de interferência no carregamento dos novos arquivos. Seguindo por esta linha, é possível ainda a escolha pela reinicialização automática da nova versão da aplicação, logo após sua atualização.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

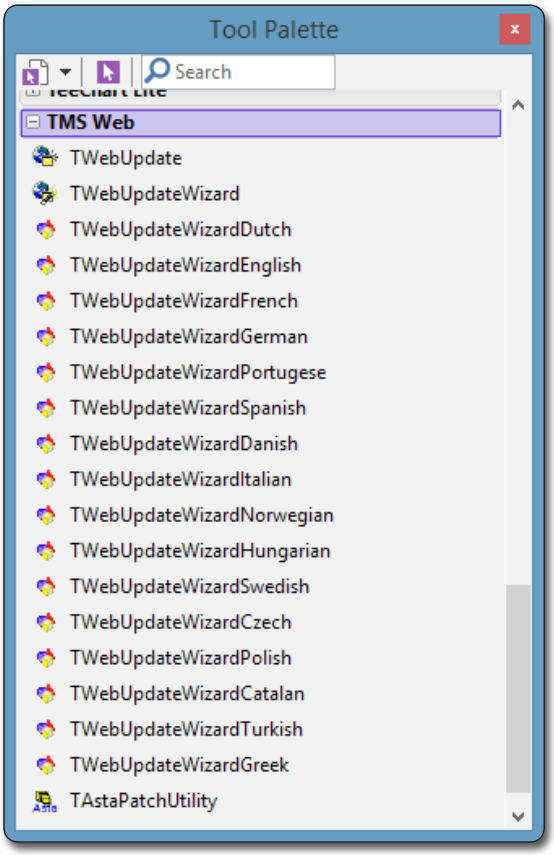

Figura 2. Tool Palette – Grupo de componentes TMS Web

Tipo de distribuição

Tendo em vista sua finalidade e essência, uma das principais configurações do componente TWebUpdate está relacionada à escolha pelo tipo de distribuição que será utilizado no processo de atualização de arquivos, neste caso a aplicação propriamente dita. Aqui, entenda o termo "tipo de distribuição" como sendo a escolha por uma das opções de transferência suportadas pelo componente: rede, FTP ou HTTP/HTTPS.

Na prática, essa definição é feita por meio da propriedade UpdateType, cujos possíveis valores se referem a um conjunto enumerado com três opções fixas – fileUpdate, ftpUpdate e http-Update – em referência a cada tipo suportado. Em termos técnicos, a primeira opção fileUpdate estabelece a transferência via rede, enquanto as duas últimas exigem invariavelmente uma conexão válida com a Internet. Neste ponto, o TWebUpdate conta com um adicional relevante, relacionado a conexões Internet obsoletas, via dial-up (discada). Para esta pontualidade, o componente provê de recursos para efetuar tentativas automáticas de conexão, livrando o usuário final de uma eventual intervenção manual. No mais, toda as tratativas internas sobre conectividade ficam concentradas na propriedade UpdateConnect, cujos valores também estão relacionados aos elementos de um conjunto enumerado que abrangem diversos cenários possíveis:

- wucConnectPromp: tenta se conectar por meio de prompt de interação com o usuário;
- wucConnectPromptHangup: tenta se conectar com prompt de interação com o usuário e provê desligamento automático;
- wucConnectSilent: tenta se conectar de forma silenciosa (sem prompt);
- wucConnectSilentHangup: tenta se conectar de forma silenciosa (sem prompt), com desligamento automático;
- wucNoConnect: sem conexão;
- wucNone: neutro.

Em complemento, Update é outra propriedade relacionada a todo este processo de transferência. De forma sucinta, sua função é controlar o comportamento do prompt citado anteriormente. Assim, dentre seus três valores possíveis, wuuSilent estabelece que o download de todos os arquivos envolvidos na atualização seja copiado de forma automática e silenciosa ao usuário. Seguindo por esta linha, wuuPromptAll determina o uso de prompt a cada arquivo a ser baixado, enquanto que wuuPromptOnce irá mostrar o prompt somente uma vez, tornando silencioso o download dos demais eventuais arquivos posteriores.

Local base de atualização

No contexto do TWebUpdate, o "local base de atualização" nada mais é do que o lugar onde estará disponível o arquivo de controle (.inf) condizente à atualização da aplicação. Tradicionalmente este local pode ser um diretório físico especificado em um servidor, um diretório virtual disponibilizado em um servidor Web, ou mesmo uma simples pasta compartilhada na rede local. Assim, para esse tipo de tratativa, o componente faz uso de sua propriedade denominada URL, que tem a incumbência de armazenar um caminho válido de local base de atualização. Tomando como exemplo um ambiente local, onde a atualização ocorrerá via rede, o seguinte caminho poderia ser utilizado como indicativo para preenchimento da propriedade URL:

```
\\grupocorp\Sistemas\arquivoDeControle.inf
```

Já para uma situação alusiva a um cenário corporativo distribuído, a atualização da mesma aplicação se daria via Internet, por meio de HTTP, o que mudaria o prognóstico da URL para o seguinte:

```
http://www.minhaempresa.com/atualizacoes/arquivoDeControle.inf
```

Atualização baseada em HTTP

Ainda sobre os contextos empresariais, é de praxe a inclusão e uso de proxy no âmbito de internet da corporação, o que acaba por exigir configurações adicionais para a devida navegação sobre o protocolo HTTP. Em vista disso, naturalmente, um eventual processo de atualização sobre esta faixa também acaba por exigir uma tratativa especial. A fim de amenizar tal "complexidade", o próprio componente toma como padrão o uso das configurações definidas no navegar Web do Windows – o Internet Explorer – descartando assim qualquer definição, por parte do desenvolvedor

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

e do usuário final, com relação às eventuais informações de proxy que possam estar relacionadas ao contexto.

Indo além, nos casos de distribuição em meio seguro, via HTTPS, onde comumente há o envolvimento de credenciais de acesso (usuário e senha), o componente trata tais informações também por meio de propriedades, nomeadas como UserID e Password, que uma vez preenchidas, servirão como base para a autenticação necessária. Adicionalmente, caso estes dados não possam ser definidos de forma fixa, seja por questões técnicas ou de política de segurança, é possível fazer com que as mesmas sejam exigidas ao usuário no instante momento de cada conexão. Novamente, isso ocorre por meio do envolvimento de outra propriedade do componente: Authenticate. Tal propriedade trabalha com um conjunto fixo de três valores possíveis, todos diretamente relacionados ao prompt de autenticação a ser mostrado, sendo waNever para nunca mostrar, waAuto para mostrar somente nos casos onde se faz necessário e waAlways para sempre mostrar, invariavelmente.

Por fim, da mesma forma que é possível se estabelecer de forma fixa as informações de usuário e senha a serem usadas na autenticação de uma alocação protegida, ProxyUserID e ProxyUserPassword são propriedades complementares que podem ser estabelecidas para uma eventual autenticação existente de proxy.

Baseado em FTP

Assim como ocorre nas atualizações baseadas em rede e HTTP, as atualizações baseadas em FTP também fazem uso da propriedade URL para a indicação do arquivo de controle de atualização. Todavia, o diferencial aqui fica por conta das definições adicionais que devem ser feitas em razão das próprias exigências naturais do contexto do protocolo. Tais definições se dão também por meio de propriedades do TWebUpdate, conforme listado a seguir:

- Host: define o nome do servidor FTP;
- Port: define o número da porta do servidor FTP;
- UserID: define o usuário de login ao servidor FTP;
- Password: define a senha de login ao servidor FTP;
- FTPDirectory: indica o diretório, localizado no servidor FTP, onde se encontra a atualização;
- FTPPassive: indica o trabalho com o modo PASV FTP.

Métodos do TWebUpdate

A plena utilização de um determinado componente se dá basicamente por um processo de duas etapas, em que a primeira consiste da configuração do mesmo, seguida de sua real utilização. Assim, com o TWebUpdate esse prognóstico não é diferente e, uma vez dada sua ideal configuração, o posterior usufruto de seus diversos métodos e eventos é que fundamentarão toda sua aplicabilidade no negócio.

Num primeiro momento, dois são os métodos tidos como essenciais ao conhecimento do desenvolvedor. O primeiro é denominado DoUpdate e, conforme seu nome indica, ele executa o processo de atualização. Além disso, assim como a grande maioria dos métodos tradicionais, ele é executado na thread principal de seu aplicativo chamador. Apesar de este poder ser considerado um comportamento padrão, tal pontualidade é ressaltada em vista do outro método a ser conhecido neste momento. Assim, DoThreadUpdate é um método bastante similar ao DoUpdate, já que possui as mesmas pretensões. O diferencial aqui fica por conta da execução do processo de atualização, que ocorre em uma segmentação (thread) separada. Em cenários mais avançados, tal comportamento tende a ser um fator positivo em questões de usabilidade, uma vez que a thread principal de execução do aplicativo estaria livre para outras atividades.

Outro ponto relevante a ser exaltado neste momento diz respeito ao encapsulamento de complexidades que o componente se propõe a fazer, tornando tudo mais facilitado em termos práticos. Como exemplo, uma simples chamada ao método DoUpdate é o suficiente para que todo o processo de atualização da aplicação seja iniciado, diferente de uma situação sem o uso do componente, o que exigiria do desenvolvedor o conhecimento e codificação de todas as tratativas necessárias ao processo.

Ainda sobre métodos, naturalmente atrelado a estes dois procedimentos, está uma função de verificação, nomeada NewVersionAvailable, que simplesmente retorna um valor booleano sobre a existência de uma nova versão da aplicação para atualização. Logo, em suma, uma combinação de uso destes métodos tidos como "essenciais", poderia ser feita da seguinte forma mostrada na **Listagem 1**.

Listagem 1. Disparando uma atualização

```
begin
  if WebUpdate1.NewVersionAvailable then
    WebUpdate1.DoUpdate;
end;
```

Eventos do TWebUpdate

Tais como métodos, diversos eventos são disponibilizados pelo componente, que são disparados durante o processo de atualização. A utilização deles acaba por prover um maior controle e monitoramento ao desenvolvedor durante a atualização. Em vista disso, a seguir são discorridos alguns dos eventos mais importantes do TWebUpdate.

OnBeforeFileDownload

Assinatura: OnBeforeFileDownload(Sender: TObject; FileIdx: Integer; FileDescription: string; var URL: string);

Evento disparado um momento antes da atualização detectada ser "baixada". Entre suas possibilidades usuais, o evento permite a alteração dinâmica da URL envolvida.

OnFileProgress

Assinatura: OnFileProgress(Sender: TObject; FileName: string; Pos, Size: Integer);

Evento disparado para a indicação de progresso do download do arquivo envolvido. Sendo assim, provê dois parâmetros,

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Pos e Size, que indicam a posição do download perante o tamanho do arquivo bem como o próprio tamanho do arquivo, respectivamente.

OnFileVersionCheck

Assinatura: OnFileVersionCheck(Sender: TObject; NewVersion, LocalVersion: string; var IsNew: Boolean);

Conforme sua nomeação propõe, este evento ocorre durante a verificação de versão de cada arquivo disponível na atualização. Na verdade, sua usabilidade está diretamente relacionada à verificação customizada (custom version check) provida pelo TWebUpdate. Em vista disso, diversos aspectos podem ser trabalhados neste instante, dado os argumentos providos pelo próprio evento:

- NewVersion: armazena a informação da nova versão, recuperada diretamente do arquivo de controle de atualização (.inf) do cenário;
- LocalVersion: faz referência à versão local de onde as informações deverão ser extraídas para comparação;
- IsNew: variável booleana que indica se a atualização atual é uma versão mais recente (ou não).

OnFileDownloaded

Assinatura: OnFileDownloaded(Sender: TObject; FileName: string);

Evento disparado assim que o download de um arquivo é concluído. Num cenário comercial, tal momento pode significar o início de diversos processos internos, desde um simples registro de Log a uma mensagem de orientação ao usuário final.

OnDownloadedWhatsNew

Assinatura: OnDownloadedWhatsNew(Sender: TObject; Text: TStrings; var Res: Integer);

Evento simples que é disparado quando há o envolvimento de um arquivo "What's New", que lista as novidades trazidas pela atualização corrente. Logo, por meio do argumento Text é possível então obter todo o conteúdo relacionado. Dentre as usabilidades deste meio, está a personalização dessas informações a fim de provê-las ao usuário final.

OnDownloadedEULA

Assinatura: OnDownloadedEULA(Sender: TObject; Text: TStrings; var Res: Integer);

Evento cuja essência se assemelha ao evento OnDownloadedWhatsNew, com o diferencial que seu disparo se dá quando do envolvimento de um arquivo EULA (**BOX 1**). Da mesma forma, todo conteúdo informativo envolvido pode ser recuperado por meio do parâmetro Text, sendo sua aceitação realizada pelo outro argumento – Res. Para este último, tal "aceitação" se deve ao fato da EULA se referir a um contrato de licença para o usuário final que, em vista da pontualidade, poderá concordar ou não com o que lhe for imposto.

BOX 1. EULA

EULA é o acrônimo para o termo em inglês End User License Agreement e sua definição se dá por um conjunto de ações permanentes que regulamentam o âmbito dos direitos autorais e de uso de um software perante o usuário final. Neste caso, "usuário final" é a denominação dada a qualquer entidade legal que venha a fazer usufruto do programa de computador em questão.

OnStatus

Assinatura: OnStatus(Sender: TObject; StatusStr: string; StatusCode, ErrCode: Integer);

Evento voltado à atualização via Web, provendo mensagens de status referente ao processo. Para tal, opera com três argumentos, sendo StatusStr para a exibição da string referente ao status, StatusCode para retornar o código de referência do status e ErrCode para indicar o tipo de erro, caso haja. Na prática, tal como já é de praxe no histórico de vida do Delphi, o valor atribuído a StatusCode está diretamente ligado a uma constante. Sendo assim, o número 0 refere-se a WebUpdateSuccess, que indica o sucesso na atualização, enquanto 2, WebUpdateNotFound, faz menção a uma atualização não encontrada. Em complemento, a seguir são listadas outras cinco constantes bastante ocasionais:

- WebUpdateAccessError = 1;
- WebUpdateNoNewVersion = 4;
- WebUpdateSignatureError = 11;
- WebUpdateWhatsNew = 12;
- WebUpdateExecAndWait = 18.

De forma similar, o número de retorno atribuído a ErrCode também faz referência a diversas constantes, tal como a seguir:

- ErrControlFileNotFound = 0;
- ErrUpdateFileNotFound = 1;
- ErrUpdateFileZeroLen = 2;
- ErrUpdateTargetEqual = 3;
- ErrUpdateSignatureError = 4;
- ErrConnectError = 5.

Arquivo de controle de atualização

Todo o fundamento das principais atividades desempenhadas pelo TWebUpdate é baseado num arquivo de controle, mais especificamente às diretrizes determinadas em seu conteúdo, que acabam por permear todo o processo de atualização. Conceitualmente, tal arquivo nada mais é do que um arquivo tradicional .INI, porém com uma extensão própria - .inf. Assim, dada sua importância, a seguir serão elencadas as seções fundamentais existentes num arquivo deste tipo. Obviamente, arquivos deste tipo podem ser plenamente elaborados de forma manual, todavia, conforme será visto mais adiante, a própria instalação do componente provê um meio alternativo e automatizado para tal, facilitando em muito a vida do desenvolvedor e do usuário final.

[update]

De forma natural, Update é a principal seção contida neste arquivo de controle de atualização, o que torna seu uso imprescindível,

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

uma vez que seus parâmetros é que servirão como base para toda a essência do processo. Na prática, duas são suas possibilidades de uso, que incidem justamente sobre as duas formas de atualização providas pelo componente: baseada em uma data ou baseada em uma versão.

Por data

De forma sucinta, no primeiro caso, a data armazenada no arquivo é comparada com a data da última atualização ocorrida no local da aplicação corrente. Sem deixar de mencionar, por padrão, o TWebUpdate persiste esta data no registro do sistema, tornando-a disponível a qualquer instante de uma eventual atualização. Logo, se a data armazenada no arquivo for maior que a data da última atualização ocorrida, o processo de update é efetivamente iniciado.

Para sua escrita, o seguinte padrão é então adotado, onde a informação "time", representativa das horas, é tida como opcional. Uma vez que "time" não é explicitamente declarada, o valor 00:00 será o utilizado de forma implícita (**Listagem 2**).

Por versão

Em complemento à atualização baseada em data está a atualização baseada em versão. Para este caso o seguinte padrão de uso é adotado (**Listagem 3**).

Listagem 2. Atualização por data

```
[update]
date=31/12/2014
time=23:00
```

Listagem 3. Atualização por versão

```
[update]
newversion=(valor)
newsize=(valor)
newchecksum=(valor)
localversion=aplicacao.exe
```

Neste caso, localversion indica o nome do arquivo da aplicação que será atualizada. Assim, tendo como base suas informações é que o TWebUpdate realizará as devidas verificações/comparações que antecedem a efetiva atualização. Logo, newversion, newsize e newchecksum caracterizam as três opções alternativas de comparação disponíveis.

De forma explicativa, newversion remete à comparação estrita de número de versão – major, minor, release e build. Já newsize e newchecksum se fundamentam na comparação do tamanho do arquivo e da soma de verificação (checksum), respectivamente.

Assim como no caso anterior, diante da opção escolhida, a sua posterioridade com relação ao arquivo da aplicação a ser atualizada, faz com que o processo de update seja iniciado.

[action]

Action é a nomeação dada a outra seção do arquivo de controle tida como fundamental neste contexto. Em resumo, sua função é armazenar uma série de parâmetros cujos valores representam ações a serem executadas em algum ponto do processo de atualização previsto. Cada parâmetro deste é então representado por uma palavra-chave, cuja as mais usuais são elencadas a seguir:

- updateURL: indica o caminho que irá atualizar a URL do arquivo de controle de atualização;
- msg: estabelece uma mensagem que será mostrada durante o processo de atualização;
- query: estabelece uma mensagem indicativa que será mostrada ao usuário possibilitando-o cancelar a atualização;
- showURL: mostra a URL envolvida no navegador;
- htmldlg: especifica um arquivo HTML que será mostrado como diálogo na URL;
- runbefore: aponta para um aplicativo qualquer que será executado imediatamente antes do início do processo de atualização;
- runafter: aponta para um aplicativo qualquer que será executado imediatamente após a transferência de todos os arquivos envolvidos na atualização.

Partindo de um ponto de vista inicial, diferentemente da seção update, action contempla parâmetros que podem ser considerados opcionais ao processo. Todavia, em cenários corporativos e mais avançados, o uso de cada um deles se torna algo natural e de suma importância às pretensões de um atualizador sistêmico. Por isso a importância do conhecimento da existência de cada um.

[WhatsNew]

De maneira opcional, o arquivo pode contar ainda com uma seção denominada WhatsNew, cuja função é indicar o arquivo externo contendo as informações sobre "o que há de novo" nesta versão do aplicativo. No cenário comercial, esse tipo de prática já é bastante usual, provendo ao usuário final a maior quantidade de informação possível sobre o que exatamente ele está recebendo naquele momento, no caso, na corrente atualização. Assim, tomando como base a atualização de uma aplicação comercial, seu conteúdo WhatsNew poderia listar então eventuais correções de "bugs" sistêmicos, bem como o acréscimo de novos recursos. Na prática, essa definição ocorre de maneira totalmente simples, uma vez que somente uma keyword está envolvida, conforme pode ser visto a seguir:

```
[WhatsNew]
file=(localização do arquivo)
```

[EULA]

Seguindo por esta linha, outra definição complementar e opcional no arquivo se dá pela inclusão da seção EULA, referente ao contrato de licença da aplicação em questão. Assim, da mesma forma que ocorre com WhatsNew, a seção EULA simplesmente

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

indica o arquivo contendo os termos do contrato em vigor, como o exemplo a seguir:

```
[EULA]
file=(localização do arquivo)
```

Além disso, outro recurso a ser destacado neste momento se relaciona tanto a WhatsNew quanto a EULA, e se faz pelo suporte a múltiplos idiomas. Em suma, cada seção especificada pode fazer referência a um arquivo em determinado idioma, tornando seu atualizador bastante abrangente. Na prática, essa atribuição trabalha em conjunto com a propriedade LanguageID do próprio componente TWebUpdate. Assim, uma vez que se é definida uma determinada linguagem em sua propriedade, o componente irá buscar por uma seção com o seguinte padrão:

```
[EULA+LanguageID] ou [WhatsNew+LanguageID]
```

Dessa forma, supondo uma abordagem tradicional em inglês (LanguageID = EN), a definição de uma seção EULA para este idioma se daria como [EULAEN], tal qual para WhatsNew ficaria [WhatsNewEN]. Por conseguinte, ainda olhando pelo lado prático, o TWebUpdate conta com uma caixa de diálogo própria que provê suporte a arquivos em formato texto (.txt e .rtf), o que acaba por contemplar os arquivos relacionados a WhatsNew e EULA.

[files]

A seção "files" é direcionada a explicitar o número de arquivos envolvidos na atualização, usando para isso a palavra-chave count, tal como é mostrado na sequência:

```
[files]
count=(número de arquivos)
```

Apesar de envolver uma definição bastante simples, a indicação do envolvimento de múltiplos arquivos em [files] tende a dar origem a outra definição complementar relacionada a cada arquivo envolvido. Logo, na ocorrência de dois arquivos no processo, a definição de ambos poderia ocorrer da seguinte maneira mostrada na **Listagem 4**.

Dentre os diversos parâmetros envolvidos, muitos deles já foram explicitados anteriormente enquanto que outros surgem como novidade. Dentre estes está a keyword filesize, que é tida como opcional e estabelece o tamanho total do arquivo, para que esta informação seja usada pelo TWebUpdate durante a indicação do progresso de download do mesmo. Já mandatory é uma opção booleana que define a obrigatoriedade do arquivo para o processo de atualização. De forma mais simples, compressed indica se o arquivo deverá ser descompactado por meio da utilização de algum dos algoritmos suportados pelo contexto. Por fim, o parâmetro hidden, também de valor booleano (0 ou 1), quando valorado com "1" estabelece a não exibição do arquivo em questão na listagem de arquivos a serem baixados durante a atualização. Todavia, seu download ocorrerá naturalmente, "escondido" do usuário.

Sem deixar de mencionar, o parâmetro "descr", conforme sua nomeação sugere, estabelece um rápido descritivo do arquivo, que será usado pelo evento OnGetFileList do componente central (TWebUpdate), em que uma de suas funcionalidades essenciais é listar os arquivos que estarão envolvidos no processo de atualização.

Listagem 4. Definição de múltiplos arquivos

```
[file1]
url=
newversion/newsize/newchecksum=
localversion=
targetdir=
descr=
compressed=
filesize=
mandatory=
hidden=

[file2]
url=
newversion/newsize/newchecksum=
localversion=
targetdir=
descr=
compressed=
filesize=
mandatory=
hidden=
```

TWebUpdateWizard

TWebUpdateWizard faz referência ao nome da classe do componente homônimo presente no agrupamento TMS Web da Tool Palette do IDE, conforme pode ser visto ainda na **Figura 2**. Por definição, sua função é fornecer um front-end (interface) amigável ao usuário, de forma a guiá-lo através de todo o processo de atualização. Em termos práticos, sua utilização está condicionada à presença de um componente TWebUpdate. Dada essa coexistência, o TWebUpdateWizard se liga ao TWebUpdate por meio de sua propriedade de mesmo nome, tal como mostrado na **Listagem 5**.

Por conseguinte, de forma simples, Execute é o nome do método que deverá ser chamado em meio a codificação, para que o assistente seja iniciado.

Listagem 5. Definição de múltiplos arquivos

```
object WebUpdate1: TWebUpdate
...
end

object WebUpdateWizard1: TWebUpdateWizard
 WebUpdate = WebUpdate1
 ...
end
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

TWebUpdateWizard Language

De forma geral, o termo TWebUpdateWizard Language faz referência aos elementos prefixados com TWebUpdateWizard, conforme listados na **Figura 3**.

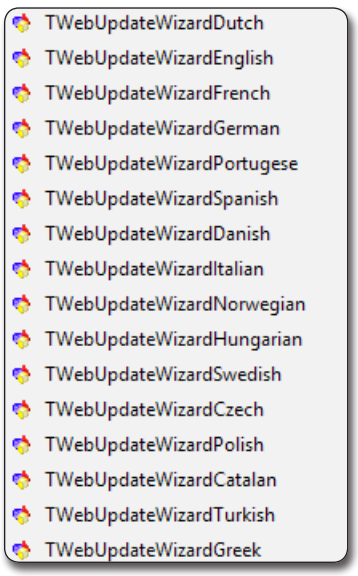

Figura 3. Componentes TWebUpdateWizard

Em suma, todos estes eles fundamentam uma mesma funcionalidade, que é a de prover a devida tradução aos termos utilizados no Wizard, atuando como uma espécie de dicionário (**Figura 4**). Sendo assim, tem o idioma como sendo seu diferencial essencial: inglês (TWebUpdateWizardEnglish), espanhol (TWebUpdateWizardSpanish), alemão (TWebUpdateWizardDutch), francês (TWebUpdateWizardFrance), etc. Essa segmentação se justifica então pela própria definição do componente, de fornecer uma interface bastante amigável e intuitiva ao usuário final. Dessa forma, de acordo com o idioma do contexto ao qual o atualizador estará envolvido, o componente relacionado deverá ser o escolhido e utilizado. Exemplificando, de forma óbvia, numa ambientação brasileira, a escolha se daria pelo componente TWebUpdateWizardPortugese, em razão do idioma português.

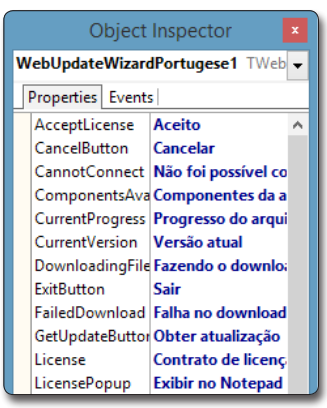

Figura 4. Propriedades do componente TWebUpdateWizardPortugese

Em seu uso prático, o componente se liga a um TWebUpdateWizard sendo indicado na propriedade Language deste.

TMS UpdateBuilder

TMS UpdateBuilder, ou simplesmente UpdateBuilder, é o nome de um aplicativo que acompanha a instalação do TMS WebUpdate, e cuja função é fazer com que a criação de arquivos controle de atualização seja mais facilitada. Isto porque, conforme já dito, tal arquivo nada mais é do que um arquivo .INI tradicional, adequado às pretensões do componente. Em vista disso, uma série de padrões devem ser seguidos, conforme as seções e keywords mostradas anteriormente, o que tende a tornar seu processo construtivo um pouco trabalhoso e repetitivo se feito de forma manual. Por esta razão, a TMS elaborou um facilitador – o TMS UpdateBuilder – que automatiza todo este processo. Uma vez aberto, o aplicativo provê a seguinte interface, conforme mostra a **Figura 5**.

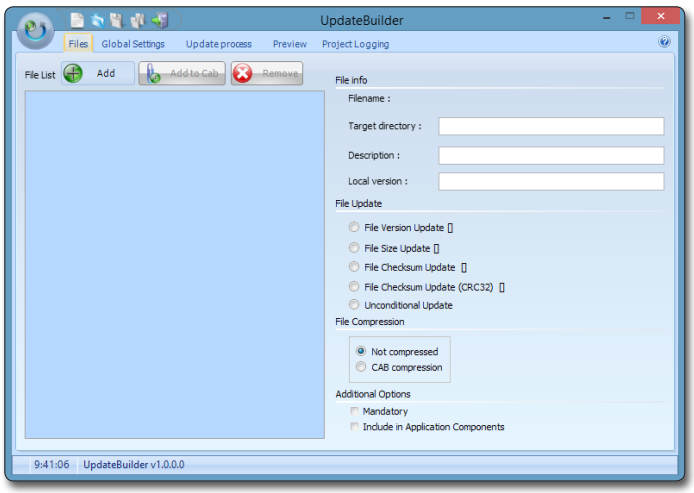

Figura 5. Interface gráfica do TMS UpdateBuilder

Em vista disso, a seguir são explanados seus principais pontos, distribuídos por entre suas cinco guias: Files, Global Settings, Update Process, Preview e Project Logging.

Guia Files

Nesta guia, o botão Add se mostra como um dos principais elementos, uma vez que é através dele que se dará a adição dos arquivos que irão compor a atualização. Partindo de uma exemplificação bastante tradicional no cenário Delphi, um cenário de esboço poderia então envolver um arquivo executável principal da aplicação (Aplicacao.exe) em conjunto com uma biblioteca modularizada (Modulo.bpl). A importância de sua usabilidade é aliada ainda ao painel da direita onde estão localizadas as seções File Info, File Update, File Compression e Additional Options, referentes a cada seção a ser determinada no arquivo de controle.

Guia Global Settings

A segunda guia, denominada Global Settings, determina algumas configurações gerais do processo, tal qual a forma de

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

verificação de versão a ser utilizada (**Figura 6**). Dentre as três opções disponíveis, "Unconditional Update" é tida como padrão, e se refere a uma atualização incondicional, sem verificações. Já "Data Based Update", conforme sua nomeação indica, faz com que o processo se baseie em uma data (e opcionalmente hora) especificada.

Por fim, o método "File Based Update" toma como base a comparação entre arquivos, determinado por um arquivo indicado na máquina local do usuário e outro equivalente no servidor.

Ainda conforme a imagem, é possível a notar a presença de diversas outras abas laterais, além da citada "Version", que acabam por fundamentar a configuração de outras seções complementares presente no arquivo.

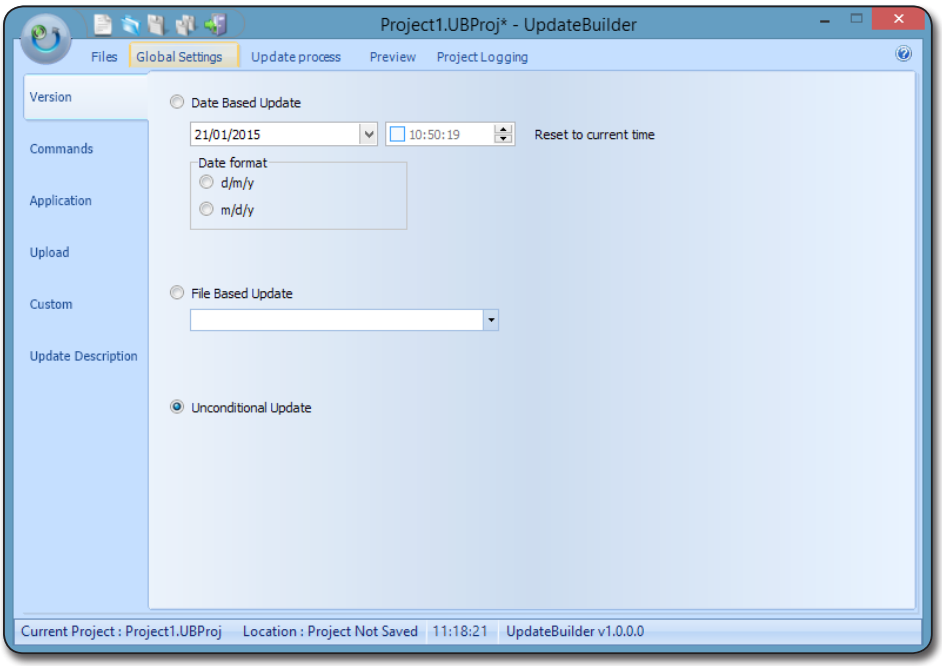

Figura 6. UpdateBuilder – Guia Global Settings

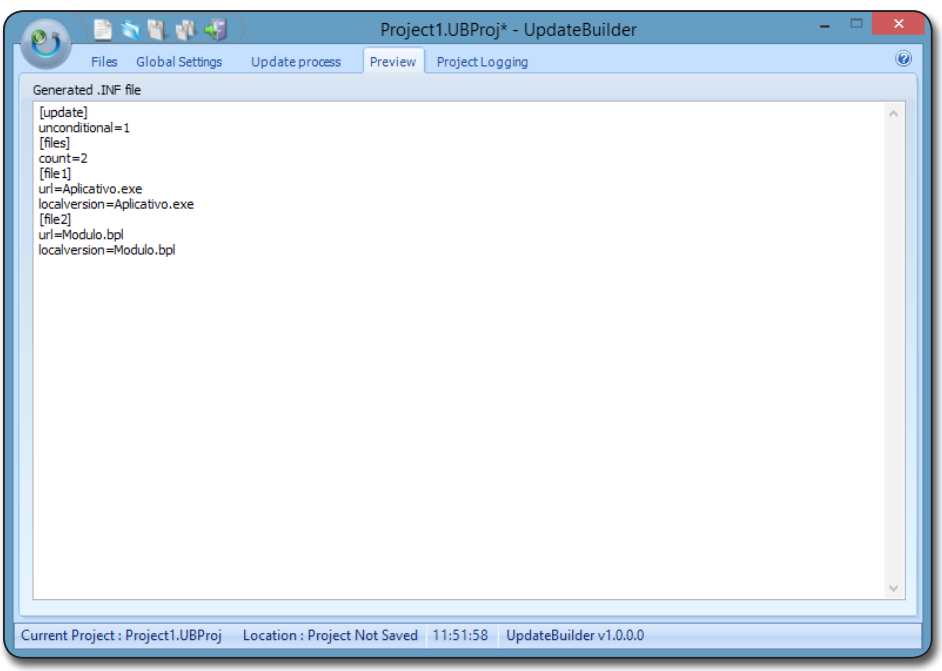

Figura 7. UpdateBuilder – Guia Preview

Guia Preview

Diante das diversas configurações impostas, a guia Preview se torna de grande valia neste contexto, uma vez que é a responsável por disponibilizar uma prévia do conteúdo do arquivo de controle que será produzido (**Figura 7**).

Guia Update process

Encerrada as parametrizações referentes ao processo de atualização, a guia Update Process pode então ser considerada a etapa final de uso do UpdateBuilder, uma vez que contempla a opção BuildProject (**Figura 8**), sob a forma de botão. Ao acioná-lo, o UpdateBuilder dá início à construção efetiva do arquivo de controle de atualização (.inf), verificando a presença e integridade de todos os parâmetros definidos e necessários ao processo. Para este fim, uma atualização real é iniciada, de forma a simular uma situação real de uso.

Guia Project Logging

A guia Project Logging nos remete às informações de log de construção do arquivo de controle, tal como exemplificado na **Figura 9.** Em seções de Log, o painel localizado nesta guia efetivamente lista os passos dados durante o build do projeto corrente, incluindo o processo de atualização realizado em conjunto. Quando da ocorrência de uma grande quantidade de informações nesta seção, o botão Clear efetua a limpeza das mensagens.

No final das contas, o uso do UpdateBuilder dá origem a um Projeto, que incidirá numa ação de Build que resultará num produto final, que é o arquivo .inf.

Exemplo de uso

A fim de estabelecer de forma definitiva todos os diversos pontos conceituais apresentados até aqui, nesta última parte do artigo será construída uma pequena aplicação de exemplo, que servirá como ponto de partida para os eventuais desenvolvimentos de cada um. Pelo fato

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

de envolver várias vertentes possíveis, a exemplificação dada aqui será a mais simples possível, mantendo o foco no uso dos conceitos essenciais do componente e suas atribuições.

Assim, seguindo por uma linha temporal, o primeiro passo a ser dado para o uso efetivo do TWebUpdate em construções Delphi, está na adição e configuração dos componentes relacionados. Tendo em vista um projeto do tipo VCL Forms, que ainda nos tempos atuais se mostra o mais usual em meio à comunidade, uma construção inicial sugerida se dá pelo uso do trio de componentes – TWebUpdate, TWebUpdateWizard e TWebUpdateWizardPortugese – dada a função já vista de cada um.

Em termos de configuração, iniciando pelo TWebUpdate, o devido preenchimento de duas de suas propriedades podem ser estabelecidas como sendo imprescindíveis em qualquer construção, tal como mostra a **Listagem 6**.

Listagem 6. Definição de múltiplos arquivos

```
object WebUpdate1: TWebUpdate
  UpdateType = fileUpdate
  URL = '[caminho]\ClubeDelphi.INF'
  ...
end
```

Nesta amostragem, fileUpdate estabelece o tipo de atualização como sendo "via rede", enquanto que a URL recebe o caminho onde estará localizado o arquivo de controle de atualização, de extensão .inf (a ser criado a seguir). De forma complementar, em TWebUpdateWizard o ajuste se dá pela configuração de suas propriedades WebUpdate e Language. Webupdate irá receber uma referência ao componente TWebUpdate e a Language deverá ser atribuído o elemento de idioma envolvido, neste caso, o TWebUpdateWizardPortugese.

Criando o arquivo de controle de atualização

Uma vez configurado os componentes envolvidos, outro fator importante para o contexto de funcionamento do TWebUpdate se dá pela criação do arquivo de controle de atualização (.inf).

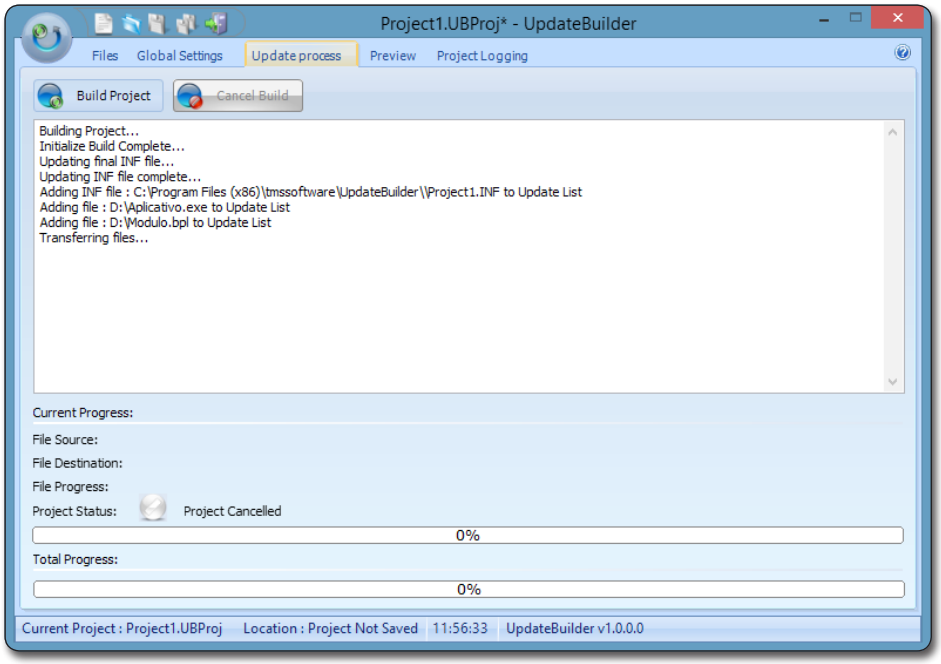

Figura 8. UpdateBuilder – Guia Preview

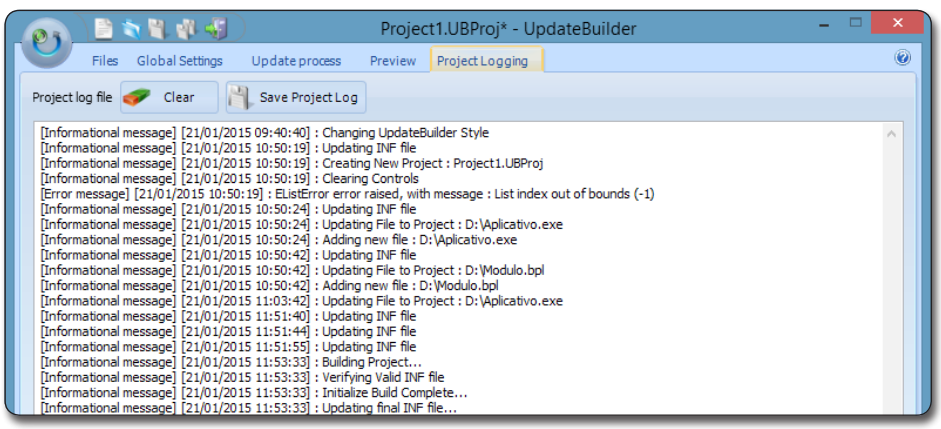

Figura 9. Update Builder – Guia Project Logging

Diante disso, a prática ideal se faz pela utilização do UpdateBuilder, já abordado anteriormente. Como exemplo de uso real, a **Listagem 7** mostra o conteúdo de um arquivo .inf produzido pelo aplicativo utilitário.

Em suma, diante das seções e keywords já explanadas anteriormente, o cenário de atualização atual é composto de um único arquivo envolvido, representativo do arquivo executável da aplicação principal (AplicacaoClubeDelphi.exe). O parâmetro unconditional (da seção update) estabelece uma atualização incondicional, sem verificações. E, por fim, na seção file1 são definidos alguns detalhamentos sobre o arquivo, tal qual a localização de sua atualização (url), o arquivo executável local (localversion) e o diretório de destino da atualização (targetdir), onde {app} refere-se ao diretório atual da aplicação.

Listagem 7. Conteúdo de exemplo de arquivo de controle

```
[update]
unconditional=1
[files]
count=1
[file1]
url=D:\Servidor\AplicacaoClubeDelphi.exe
localversion=AplicacaoClubeDelphi.exe
targetdir={app}
descr=Aplicação Clube Delphi
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Iniciando o processo

Por questões óbvias, a opção pelo uso do componente de Wizard, implica no direcionamento do processo ao assistente, que irá guiar o usuário final por todo o processo de atualização. Diante do cenário que se apresenta, a simples chamada ao método Execute do componente relacionado é o suficiente para que o Wizard seja iniciado. Sendo assim, a seguir é mostrada a referida chamada ao método por meio do evento OnClick de um botão simples (TButton), como mostra a **Listagem 8**.

Indo além, na sequência, a **Figura 10** mostra a tela inicial do assistente, que já se encontra em execução.

Listagem 8. Conteúdo de exemplo de arquivo de controle

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  WebUpdateWizard1.Execute;
end;
```

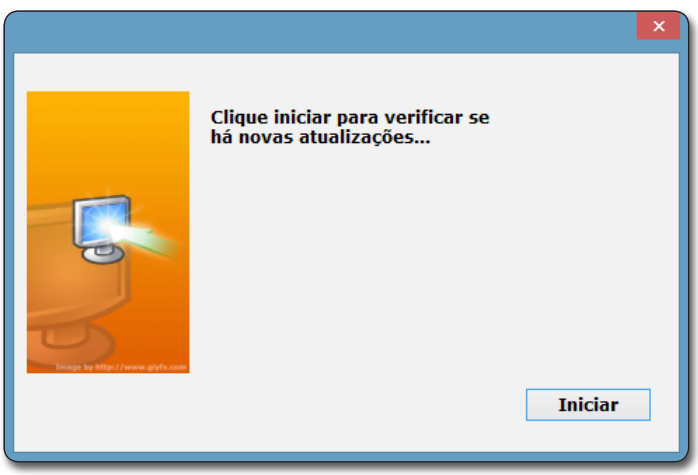

Figura 10. Tela inicial do Wizard de atualização

O clique sobre o botão Iniciar efetivamente dispara o processo, cuja etapa emergente é representada pela verificação da existência de uma "versão" para atualizar. Uma vez que é encontrada, um novo passo é mostrado ao usuário final, conforme ilustrado na **Figura 11**.

Ao clicar em "Obter atualização", o assistente então se encarrega em listar, no passo seguinte, os arquivos disponíveis para download. Para esta exemplificação, apenas um está apresente, conforme definido no arquivo de controle (**Figura 12**).

Por fim, uma vez iniciada a atualização, uma mensagem indicativa da finalização do processo é então exibida ao usuário no passo final do assistente (**Figura 13**).

Após vermos na prática a aplicação de uma atualização automática, fica comprovada a importância de tais elementos em qualquer cenário produtivo de desenvolvimento de software, ao qual o Delphi está inserido. A utilização do componente TWebUpdate se justifica pela razão principal do encapsulamento de todas as eventuais complexidades que estão atreladas ao contexto.

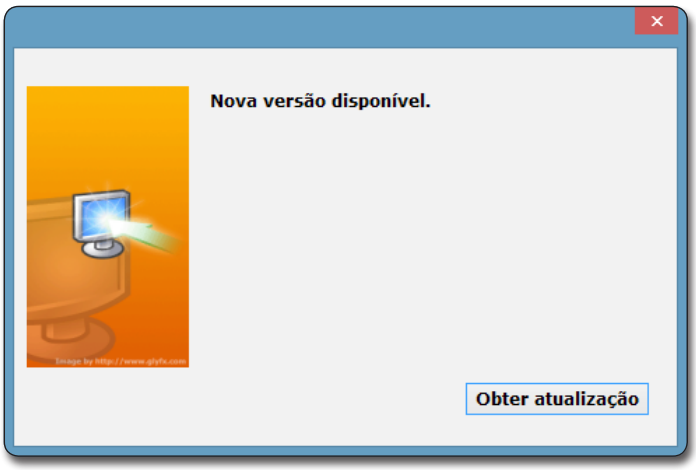

Figura 11. Wizard de atualização – nova versão disponível

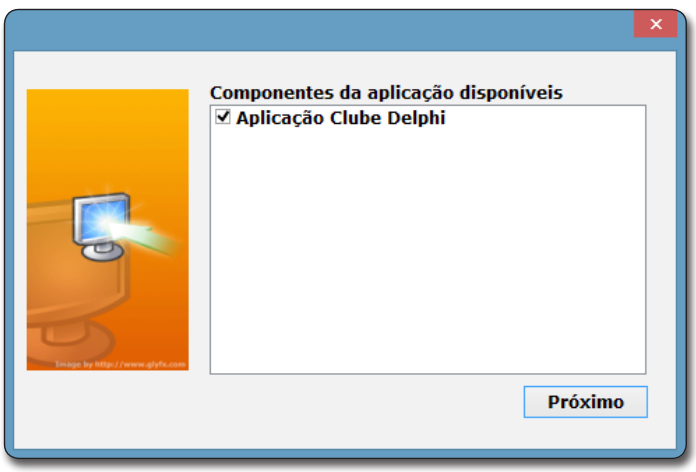

Figura 12. Wizard de atualização – selecionando o arquivo para download

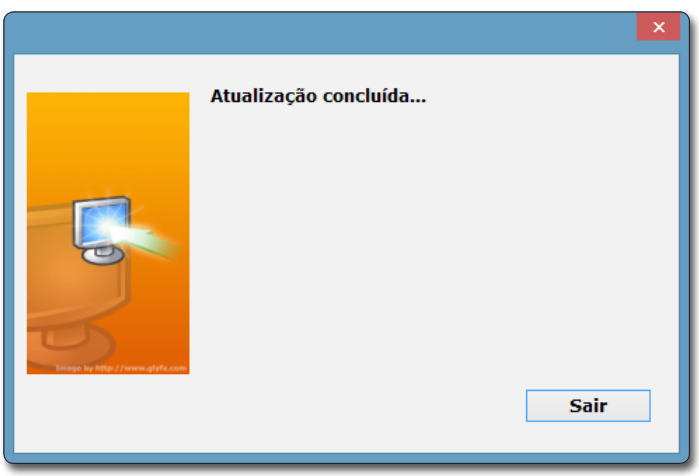

Figura 13. Wizard de atualização – mensagem de sucesso de conclusão

Adicionalmente, somado a isso está também todo o dinamismo RAD que os componentes tendem a prover ao desenvolvedor, por meio de suas configurações visuais e manuseamento físico, já habitual e de excelência no IDE desde sua concepção.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Por conta disso, a cada novo componente conhecido ou apresentado, seja este gratuito ou de cunho comercial (pago), um novo desafio é lançado ao desenvolvedor, dada a invariável necessidade no aprofundamento do tema, para uma ideal utilização posterior. No caso do TWebUpdate, a partir da introdução feita por este artigo, a "complexidade" posterior se dá pelo ideal ajuste do elemento ao cenário ao qual ele será inserido.

Links:

RAD Studio XE7 – Página oficial do produto
http://www.embarcadero.com/products/rad-studio

RAD Studio XE7 – Download Trial
https://downloads.embarcadero.com/free/rad_studio

TWebUpdate – Página oficial do produto
http://www.tmssoftware.com/site/wupdate.asp

Autor

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ e pós-graduado em Engenharia de Componentes. Atua como Analista Programador Delphi há 9 anos.

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

CURSOS ONLINE

A Revista Java Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

CONHEÇA ALGUNS DOS CURSOS:

- Curso de noSQL (Redis) com Java
- Curso Básico de JDBC
- Java Básico: Aplicações Desktop
- JSF com Primefaces
- Conhecendo o Apache Struts

Para mais informações :
www.devmedia.com.br/curso/javamagazine
(21) 3382-5038

DEVMEDIA

Desenvolvendo um Sistema Financeiro em Delphi – Parte 2

Este artigo faz parte de um curso

O paradigma orientado a objetos ao contrário do que muitos pensam surgiu antes do paradigma de programação estruturada, porém se popularizou apenas nos anos 90.

Aos poucos foi se notando as várias vantagens que a orientação a objetos traz para os sistemas de informação e as linguagens que adotam este paradigma, como Delphi, Java, C#, foram se popularizando e tornando-se cada vez mais utilizadas pelos desenvolvedores.

Uma das maiores vantagens da programação orientada a objetos é o fato de ela facilitar muito posteriormente a manutenção de um sistema, porque as classes e objetos ficam encapsulados e podem ser substituídos separadamente uns dos outros, evitando os ajustes em cascata. Neste sentido quanto maior a coesão (**BOX 1**) de uma classe, maior a facilidade de se fazer manutenção no sistema.

BOX 1. Coesão

Coesão é um princípio da orientação a objetos que não é novo, porém muitas vezes não levado em consideração pelos desenvolvedores. É o fato de uma classe ter apenas uma responsabilidade bem definida, para que tenhamos uma arquitetura robusta e um bom design de software. Quanto mais responsabilidades tiver uma classe, maior o impacto que ela vai causar no sistema e consequentemente mais dificultoso será sua manutenção.

Uma desvantagem que podemos citar da orientação a objetos é o fato de um projeto bem elaborado ter um tempo de desenvolvimento maior em relação a outros paradigmas, como o estruturado ou orientado a eventos. As maiores vantagens acabamos percebendo a longo prazo e não a curto prazo.

Fique por Dentro

Continuando com o desenvolvimento de um sistema financeiro, neste artigo abordamos a criação do banco de dados com o Firebird 2.5, onde uma das principais tabelas da regra de negócio é criada. Também abordamos as diferenças nos paradigmas de programação e detalhamos a Orientação a objetos, explicando seus conceitos. Também é a apresentada uma classe que centraliza a conexão com o banco de dados e aplica o padrão singleton.

Paradigmas de Linguagens de Programação

Antes de entrarmos no mundo da programação orientada a objetos, precisamos conhecer um pouco sobre os paradigmas das linguagens de programação de computadores, paradigmas estes que até os tempos atuais ainda dividem espaço junto à comunidade desenvolvedora de software, buscando a resolução de problemas no mundo real, criando soluções para os problemas de forma computadorizada, através do produto de software.

Antes da programação do software, o problema a ser revolvido deve ser analisado e transformado em uma documentação contendo aspectos fundamentais ao domínio do negócio e de sua solução, documentação esta que posteriormente servirá como base para desenvolver o produto de software, ou sistema computacional, que tenha como objetivo principal a resolução do problema central e assim possa cumprir com seu papel, atendendo a todas as necessidades a que se propõe.

Mas afinal, o que é o tal paradigma? Podemos conceituar um paradigma como uma visão ou um ponto de vista do mundo real (da realidade em que vivemos) e a forma de atuação sobre tal concepção. Resumindo, é a forma como o analista e o programador lidam com um determinado problema na busca de uma solução em forma de sistema de software.

Os paradigmas são classificados em imperativo, estruturado, funcional, lógico e orientado a objetos. Cada qual possui seus conceitos e métodos de como abordar um determinado problema do mundo real para elaborar e propor uma solução para o mesmo.

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Há seguir conheceremos um pouco sobre os paradigmas das linguagens de programação:

• **Paradigma Imperativo:** o paradigma imperativo foi o primeiro paradigma a existir. Também é conhecido como paradigma procedural e trata de resolver um problema com comandos (instruções) um após o outro, até sua solução, ou seja, de forma sequencial. A arquitetura de computadores exerce uma forte influência para projetar uma linguagem de programação, assim sendo, as linguagens consideradas imperativas têm como base a tão conhecida arquitetura de computadores de Von Neumann. A linguagem de programação imperativa tem algumas características como: as variáveis que reservam os espaços na memória, comandos para atribuição e transferência de dados, execução sequencial das instruções e a possibilidade de repetição de blocos de instrução. Podemos considerar como linguagens imperativas o ALGOL, Assembler, Basic, Pascal, Cobol, C, Fortran e etc.

• **Paradigma Estruturado:** este paradigma é caracterizado pela forma de programação que deve ser composta por três estruturas simples: sequência, decisão e iteração. Na prática, é uma programação modular, onde a estrutura de um programa deve ser simples e usar funções e sub-rotinas. Cobol, Pascal, Basic e C são exemplos de linguagem de programação estruturada.

• **Paradigma Funcional:** o propósito deste paradigma é dividir um problema em partes menores (funções) que recebem dados de entrada e retornam dados nas saídas para a base chamadora. Dessa forma, este paradigma busca na programação resolver o problema central através da divisão do problema em partes menores e, ao final do processamento, mostrar o resultado. A linguagem LISP é baseada no modelo funcional.

• **Paradigma Lógico (declarativo):** para definir este paradigma precisamos fazer uma pergunta que o precede: Qual o problema? O mesmo trata de descobrir um algoritmo geral para resolução de um determinado problema em questão. Prolog, Planner e Popler são exemplos de linguagens que seguem o paradigma lógico declarativo.

• **Paradigma Orientado a Objetos:** no paradigma da orientação a objetos, tratamos o problema como um conjunto de objetos, ou seja, quando temos um problema do mundo real a resolver, primeiro buscamos abstrair todas as características e aspectos do domínio do negócio e assim transformar em um conjunto de objetos que irá fazer parte da solução do problema central. Não necessariamente esses objetos precisam ser algo concreto, ou seja, podemos ter também objetos abstratos. A venda de mercadorias é exemplo disso, onde temos os produtos que são objetos concretos, que podemos pegar e tocar, já o pedido de venda que também é um objeto, porém algo abstrato que não podemos tocar, mas o mesmo existe dentro do contexto a ser resolvido. Na Orientação a Objetos tudo é um objeto, ou seja, a estrutura do software é dividida em unidades que denominamos de classes. As classes, por sua vez, são uma forma de molde composto de atributos e métodos para criar objetos e representar um conjunto deles. Todo objeto tem um estado que é defino pelos os valores contidos nos seus atributos e também tem comportamentos que são definidos pelos seus métodos. Neste contexto da orientação a objetos, o software tem vários objetos que interagem entre si por meio de troca de mensagens. Um objeto envia uma mensagem a outro objeto fazendo a chamada de alguma operação contida em um de seus métodos, também sendo possível recuperar e alterar os valores contidos em seus atributos.

Uma observação importante é que uma determinada linguagem de programação pode ter como base de sua criação o uso de mais de um paradigma.

Um paradigma pode estar presente em diversas etapas do desenvolvimento do software, como na análise, no projeto e na programação.

Orientação a Objetos: Abstração

A abstração é uma tarefa importantíssima na orientação a objetos, pois durante o processo de análise do domínio do problema a ser resolvido, é importante saber separar as coisas relevantes ao projeto das que não pertencem ao contexto para o desenvolvimento do software.

Mas afinal, o que é abstração? Ao procuramos no dicionário, obteremos uma reposta que define abstração como sendo o ato de abstrair, ou seja, isolar mentalmente partes de um todo para considerar as partes de representação que não são dadas separadamente na realidade.

O ato de abstrair é um processo onde temos uma visão da realidade e precisamos montar uma visão mentalmente abstrata dos aspectos importantes ao contexto da solução que se pretende construir, através do uso do substantivo abstrato para criar objetos que podem ter ações e estados.

Na orientação a objetos, quando falamos em abstração, abstrair objetos do domínio do problema, deve-se obter como retorno um modelo conceitual para auxiliar na resolução do problema, modelo este que pode ser representado de formas diferentes nas etapas de desenvolvimento do software.

A abstração não é uma tarefa fácil quando se trata de desenvolvimento de software, pois as pessoas pensam de formas diferentes. Então, ao realizar uma representação mental de algo, uma pessoa pode obter uma representação para uma solução diferente da de outra pessoa, analisando o mesmo contexto do problema.

Este processo da criação do modelo conceitual inicial para o desenvolvimento de um software deve ser tratado cuidadosamente, e se possível, detalhar minuciosamente para que fique claro o contexto a ser trabalho no projeto, pois quando realizamos uma abstração fora do domínio do problema central, podemos estar cometendo um erro de passar informação à equipe de desenvolvimento do software que não condiz com a realidade e não irá atender as reais necessidades do cliente.

Quando abstraímos algo e o transformamos em um objeto, estamos realizando duas formas de abstração. Fazemos a abstração de dados, que forma as propriedades do objeto definido pelos seus atributos, e também realizamos abstração de procedimentos, que forma as operações que são expostas através dos métodos de um objeto, ou seja, os serviços disponíveis pelo objeto que formam um conjunto de ações que o mesmo pode executar.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Orientação a Objetos: Encapsulamento

O encapsulamento é o ato de ocultar e proteger informações e a implementação das ações que um objeto pode realizar, de forma a expor uma interface pública através de seus métodos, para que outros objetos possam acessar e fazer uso dos seus recursos, mas de forma controlada e organizada.

Anteriormente, quando falamos sobre a abstração, foi mostrado o processo do Gap Semântico, onde analisamos o problema do mundo real e transformamos as entidades em objetos, somente com as características inerentes ao software a ser desenvolvido.

A partir daí construímos uma solução para o problema através de produto de software, que é composto por um conjunto de classes que serve como molde para criar objetos em tempo de execução. Esses objetos se comunicam através da troca de mensagens, realizando chamadas aos recursos uns dos outros para realizar algo, mas o objeto que chama um método não precisa saber como tal tarefa é realizada, pois só interessa que a mesma seja executada.

Neste ponto temos ações disponíveis através de uma interface (métodos públicos), mas com as implementações de tarefa encapsulada, ou seja, oculta aos agentes externos. Podemos tomar como exemplo dois objetos "caixa" e "venda", onde o objeto caixa precisa realizar uma venda através do método "Realizar venda" do objeto venda. Neste momento o objeto caixa não precisa saber como uma venda é feita internamente, pois quem realiza a venda de fato é o objeto venda, que tem implementado em seu método "Realizar venda" os passos necessários para executar essa ação.

Podemos concluir que o encapsulamento tem enormes vantagens, pois podemos agrupar vários atributos e métodos em uma classe para um determinado propósito, sem que seu funcionamento interno seja exposto.

Orientação a Objetos: Herança

A herança é um elemento primordial para o paradigma de programação orientada a objetos, pois com seu uso podemos criar uma nova classe que herde características de uma classe já existente no contexto da aplicação.

Este conceito de reutilização de componentes de software, já existente em vários novos projetos, é mais uma das características fortes da programação orientada a objetos, pois nem sempre é necessário empregar tempo no desenvolvimento de um recurso que já existe e pode ser reutilizado e adaptado.

Quando utilizamos o recurso da herança, podemos poupar várias linhas de código em relação a outros paradigmas de programação. A herança nos possibilita criar uma hierarquia entre as classes: quando herdamos características de uma classe, essa passa a ser denominada classe pai (classe base), e a classe que recebe a herança é denominada classe filha (classe estendida), que se especializa em relação à classe base para realizar uma tarefa específica.

Orientação a Objetos: Polimorfismo

O conceito de polimorfismo trata de algo tomar várias formas. Se verificarmos a formação da palavra, iremos encontrar a estrutura de "poli", que significa múltiplas, e "morfo" que se significa forma, representando algo que em determinado momento pode assumir formas diferentes de acordo com a necessidade.

Na POO (Programação Orientada a Objetos) o polimorfismo está ligado diretamente à comunicação entre os objetos, na forma como eles trocam as mensagens e como o objeto receptor deve reagir ao receber a mensagem. Considerando que o objeto emissor não conhece o funcionamento interno do objeto receptor, cabe a esse segundo decidir qual é a melhor forma de responder a uma mensagem, dependendo da forma como ela foi recebida.

Assim, chamadas a um mesmo método podem resultar em resultados diferentes, dependendo do contexto em que a ação foi executada.

Também identificamos a presença do polimorfismo em situações onde há herança entre classes. Um objeto que for planejado a partir da estrutura de uma classe pai, poderá se comportar de formas diferentes se for criado usando uma classe que herde dessa classe original.

Por exemplo, uma classe pai implementa determinado método, mas suas classes filhas, quando herdam esse método, podem definir que seu funcionamento será completamente diferente do original e das demais classes filhas, especializando uma ação para um fim específico.

Banco de Dados Firebird

O Banco de Dados utilizado no desenvolvimento deste artigo é o Firebird, na sua versão 2.5, que é um banco de dados muito rápido, gratuito e bastante utilizado por desenvolvedores Delphi desde o seu lançamento. Já existe uma versão 3.0 deste banco de dados, porém ainda está em fase de testes, por isso utilizaremos ainda a versão 2.5.

O Firebird nos oferece uma ferramenta para a criação e manipulação de bancos de dados chamada ISQL, porém esta é disponibilizada na forma de linha de comando e por isso faz-se necessário o uso de outra ferramenta para trabalharmos com ele, esta é o IBExpert, que oferece muitos recursos que facilitam muito no dia a dia da administração de bancos de dados Firebird, por isso também será usado neste artigo.

Acionando o menu Database/Create Database nos deparamos com a **Figura 1**, onde configuramos várias informações referentes ao banco de dados a ser criado. A seguir uma descrição detalhada de cada uma destas opções, para alguns podem ser informações já sabidas, mas para quem está iniciando são indispensáveis para entendermos como funciona um banco de dados:

- Server: traz as opções Local e Remote, sendo que a opção remota nos possibilita fazer o acesso via rede;
- Server name: neste campo informamos qual o número IP do nosso servidor de banco de dados, neste caso 127.0.0.1 indicando que será a própria máquina;
- Protocol: aqui informamos qual será o protocolo de comunicação que será utilizada com o servidor de banco de dados. Geralmente é utilizado o mais conhecido TCP/IP, mas ainda temos outros;
- Database: nesta opção indicamos qual será o caminho da base de dados;

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

• Client Library File: arquivo da biblioteca cliente que será usado para fazer a comunicação da aplicação cliente e o banco de dados;
• Username/Password: usuário e senha do banco de dados;
• Page Size: tamanho da página do banco de dados, deve-se deixar o mesmo tamanho do cluster do HD, deixando o banco mais performático;
• Charset: define como o Firebird traduzirá um caractere em seu respectivo valor numérico, na tabela de caracteres e vice-versa e quais símbolos estarão disponíveis para serem utilizados nos campos string. Para o Brasil é interessante utilizar o WIN1252, pois permite caracteres acentuados;
• SQL Dialect: dialeto três é o mais recente e compatível com bancos de dados nas suas versões mais novas;
• Collation: para este charset, o padrão no Brasil é utilizar o WIN_PTBR que permite caracteres acentuados. A opção de escolha do Collation está disponível apenas a partir da versão 2.5 do Firebird. Nas versões anteriores o próprio banco de dados definia esta opção de forma automática de acordo com o Charset escolhido.

Logo após a criação do banco de dados, fazemos o registro do mesmo no IBExpert, conforme a **Figura 2**, onde devemos atentar para alguns itens importantes:
• Server Version: aqui informamos a versão do Firebird que estamos fazendo uso, no caso a versão 2.5;
• Database Alias: definimos um alias para que seja facilitado o acesso à base de dados;
• Font Characters Set: é a configuração da fonte das letras utilizadas no IBExpert, o ideal é utilizar o padrão ANSI_CHARSET.

Tabela de Origens de Pagamentos e Recebimentos

A primeira tabela de nosso sistema (**Listagem 1**) será onde ficará guardadas as origens de pagamentos e recebimentos. Estas origens servirão tanto no módulo de contas a pagar quanto no módulo de contas a receber temos a figura das origens de pagamentos e recebimentos. Essas origens funcionam como uma categoria para cada lançamento que for efetuado, tanto a receber quanto a pagar e ajudarão muito na hora de buscarmos lançamentos específicos e também na hora de criarmos relatórios.

Tabela de Contas a Pagar e Receber

Para as contas a pagar e receber faremos uso de uma única tabela chamada CONTAS, onde esta terá um campo chamado ID_REFERENCIA que irá apontar para um cliente quando for conta a receber e para um fornecedor quando a conta for a pagar, conforme **Listagem 2**.

Tabela Itens da Conta

Cada conta, tanto a pagar quanto a receber poderá ter um ou mais itens que ficarão registrados na tabela ITENS_CONTAS, da **Listagem 3**. Cada parcela poderá conter uma data de vencimento e o valor. O número de documento servirá para identificá-la na hora do pagamento. O campo observação poderá ser utilizado de forma livre, afim de registrar outras informações que se julgar importante.

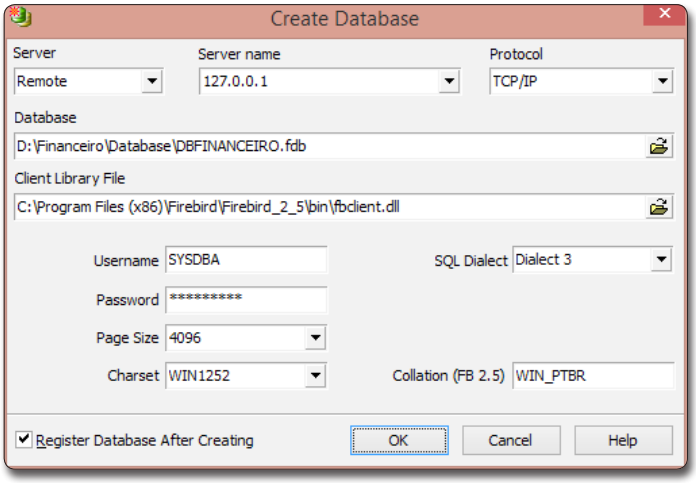

Figura 1. Criando o Banco de Dados no IBExpert

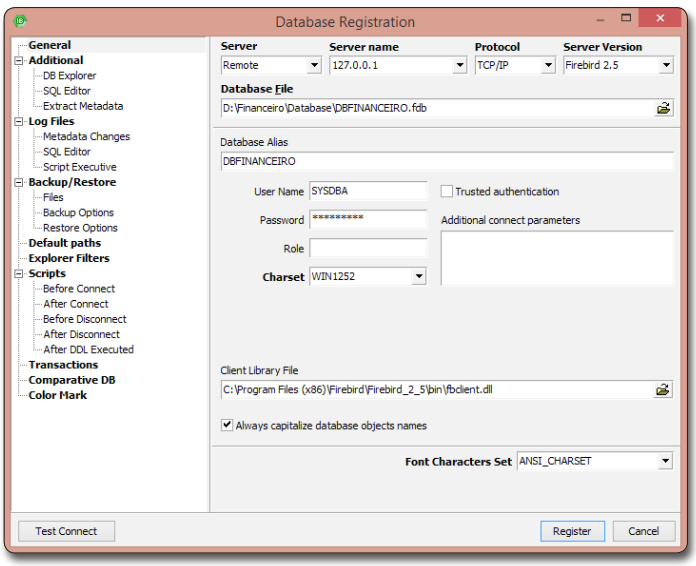

Figura 2. Criando o Banco de Dados no IBExpert

Tabelas de Clientes e Forncedores

Temos ainda clientes e fornecedores, que serão duas tabelas bastante simples e que estarão relacionadas à tabela de CONTAS. O que difere uma tabela da outra é o documento, na tabela CLIENTES temos a figura do CPF do cliente, já na tabela FORNECEDORES temos o documento CNPJ, conforme pode ser observado na **Listagem 4**.

FireDAC

Nos exemplos contidos neste artigo faremos uso da nova engine de acesso a dados da Embarcadero, o FireDAC, antigo AnyDac.

O FireDAC possui diversas vantagens e recursos interessantes como Live Data, Array DML, API Unificada, Tratamento de Erros Unificado, Arquitetura Simplificada, Restauração Automática de Conexão com a Base de Dados e Suporte a codificações ANSI e Unicode.

São um conjunto de componentes de alta performance, de fácil utilização e provê conexão com vários bancos de dados, tanto locais quanto coorporativos. Cada banco de dados possui um

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

driver e o FireDAC trabalha com cada banco de forma diferente. Os drivers do FireDAC são nativos para cada banco de dados e ainda possui pontes para ODBC e dbExpress. Para quem tem projetos multicamadas com Delphi e DataSnap, ou também servidores REST, basta migrar a parte Server de DBExpress para FireDAC, pois o TFDQuery é um TDataSet, portanto compatível com o TDataSetProvider e TClientDataSet. Na parte Client, continua o TClientDataSet sem necessidade de qualquer alteração.

Listagem 1. Tabela ORIGENS_PAGAMENTO

```
01 create table ORIGENS_PAGAMENTO (
02    ID_ORIGEM integer not null,
03    DESCRICAO varchar(100) not null,
04    TIPO_ORIGEM char(1) not null check('P','R')
05 );
06 alter table ORIGENS_PAGAMENTO add constraint PK_ORIGENS_PAGAMENTO
07    primary key (ID_ORIGEM);
08 create sequence GEN_ORIGENS_PAGAMENTO_ID;
09 create trigger ORIGENS_PAGAMENTO_BI for ORIGENS_PAGAMENTO
10 active before insert position 0
11 as
12 begin
13    if (new.ID_ORIGEM is null) then
14       new.ID_ORIGEM = gen_id(GEN_ORIGENS_PAGAMENTO_ID, 1);
15 end
```

Listagem 2. Tabela CONTAS

```
01 create table CONTAS (
02    ID_CONTA integer not null,
03    IND_TIPO_CONTA char(1) not null check('P','R'),
04    VALOR_TOTAL numeric(15, 2),
05    DATA_LANCAMENTO date,
06    NUM_PARCELAS integer,
07    IND_SITUACAO char(1) check ('A','P','C'),
08  ID_REFERENCIA integer,
09    OBSERVACAO varchar(500)
10 );
11 alter table CONTAS add constraint PK_CONTAS
12    primary key (ID_ORIGEM);
13 create sequence GEN_CONTAS_ID;
14 create trigger CONTAS_BI for CONTAS
15 active before insert position 0
16 as
17 begin
18    if (new.ID_CONTA is null) then
19       new.ID_CONTA = gen_id(GEN_CONTAS, 1);
20 end
```

Listagem 3. Tabela ITENS_CONTAS

```
01 create table ITENS_CONTAS (
02    ID_ITEM_CONTA integer not null,
03    NUM_DOCUMENTO varchar(15) not null,
04    VALOR numeric(15, 2),
05    DT_VENCIMENTO date,
06    OBSERVACAO varchar(500)
07 );
08 alter table ITENS_CONTAS add constraint PK_ITENS_CONTAS
09    primary key (ID_ITEM_CONTA);
10 create sequence GEN_ITENS_CONTAS_ID;
11 create trigger ITENS_CONTAS_BI for ITENS_CONTAS
12 active before insert position 0
13 as
14 begin
15    if (new.ID_ITEM_CONTA is null) then
16       new.ID_ITEM_CONTA = gen_id(GEN_ITENS_CONTAS, 1);
17 end
```

Listagem 4. Tabelas CLIENTES e FORNECEDORES

```
01 create table CLIENTES (
02    ID_CLIENTE integer not null,
03    NOME_CLIENTE varchar(50) not null,
04    CPF varchar(11)
05 );
06 alter table CLIENTES add constraint PK_CLIENTES
07    primary key (ID_CLIENTE);
08 create sequence GEN_CLIENTES_ID;
09 create trigger CLIENTES_BI for CLIENTES
10 active before insert position 0
11 as
12 begin
13    if (new.ID_CLIENTE is null) then
14       new.ID_CLIENTE = gen_id(GEN_CLIENTES_ID, 1);
15 end;
16
17 create table FORNECEDORES (
18    ID_FORNECEDOR integer not null,
19    NOME_FORNECEDOR varchar(50) not null,
20    CNPJ varchar(14)
21 );
22 alter table FORNECEDORES add constraint PK_FORNECEDORES
23    primary key (ID_FORNECEDOR);
24 create sequence GEN_FORNECEDORES_ID;
25 create trigger FORNECEDORES_BI for FORNECEDORES
26 active before insert position 0
27 as
28 begin
29    if (new.ID_FORNECEDOR is null) then
30       new.ID_FORNECEDOR = gen_id(GEN_FORNECEDORES_ID, 1);
31 end;
```

Tem suporte a Firemonkey e VCL e possui um conjunto de componentes visíveis e não-visíveis, DataSets, Adapters. O FireDAC é um framework que possui um conjunto de classes para o trabalho e componentes TDataSet, semelhante aos componentes do BDE por exemplo. O FireDAC possibilita o desenvolvimento de aplicações para múltiplos banco de dados de forma RAD e rápida, algo que o DBExpress tentou fazer, mas era bem complicado, pois tínhamos de que fazer várias configurações como, por exemplo, o mapeamento de campos TFields de acordo com cada banco utilizado. Uma das principais vantagens do FireDAC é a sua alta performance de acesso e a enorme quantidade de recursos e configurações disponíveis.

Criando uma Classe de Conexão com o Banco de Dados

Antes de começarmos a desenvolver as classes DAO, primeiramente devemos desenvolver uma classe de conexão (**Listagem 5**) que faça acesso à base de dados, para que possamos fazer comandos de consulta e atualização no banco de dados. Essa classe é implementada utilizando o padrão Singleton (**BOX 2**), para que tenhamos apenas uma única instância dela.

Em nosso exemplo definimos diretamente o endereço 127.0.0.1 (localhost) como máquina que será o servidor de banco de dados, para facilitar a conexão, mas ao estarmos desenvolvendo um sistema real, devemos isolar esta informação em um arquivo externo, como, por exemplo, um arquivo .INI ou .XML, para que possamos modificar os dados de conexão sem termos que recompilar o aplicativo.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

BOX 2. Singleton

O padrão Singleton permite criar objetos únicos para os quais há apenas uma instância. Este padrão oferece um ponto de acesso global, assim como uma variável global, porém sem as desvantagens das variáveis globais. Para entendermos e usarmos bem o padrão de Projeto Singleton é necessário apenas dominar bem as variáveis e métodos de classe estáticos além dos modificadores de acesso.
O Padrão Singleton tem como definição garantir que uma classe tenha apenas uma instância de si mesma e que forneça um ponto global de acesso a ela. Ou seja, uma classe gerencia a própria instância dela além de evitar que qualquer outra classe crie uma instância dela. Para criar a instância tem-se que passar pela classe obrigatoriamente, nenhuma outra classe pode instanciar ela. O Padrão Singleton também oferece um ponto global de acesso a sua instância. A própria classe sempre vai oferecer a própria instância dela e caso não tenha ainda uma instância, então ela mesma cria e retorna essa nova instância criada.

Listagem 5. Classe de Conexão

```
01 unit uConnection;
02 interface
03 uses
04  FireDAC.Comp.Client, FireDAC.Stan.Def, FireDAC.Phys.MSSQL,
    FireDAC.Stan.Error,
05  FireDAC.Stan.Async, FireDAC.VCLUI.Wait, FireDAC.DApt;
06 type
07  TConnection = class
08  private
09   class var FDConnection: TFDConnection;
10  public
11   class function GetConnection: TFDConnection;
12  end;
13 implementation
14 { TConnection }
15 class function TConnection.GetConnection: TFDConnection;
16 begin
17  FDConnection := TFDConnection.Create(nil);
18  FDConnection.Params.DriverID := 'FB';
19  FDConnection.Params.Database := '127.0.0.1:C:\Artigo\';
20  FDConnection.Params.UserName := 'SYSDBA';
21  FDConnection.Params.Password := 'masterkey';
22  FDConnection.Connected := True;
23  Result := FDConnection;
24 end;
25 end.
```

Impedância Objeto Relacional

A Orientação a Objetos trouxe inúmeras vantagens em relação a programação estruturada, como a representação de objetos de maneira bastante semelhante ao mundo real. Porém isto torna-se um desafio a mais quando precisamos fazer a persistência destes objetos nos bancos de dados relacionais.

Existem soluções de bancos de dados orientados a objetos, mas estes ainda não se tornaram populares, sendo os bancos de dados relacionais ainda muito mais utilizado que este.

Não existe uma conversão direta entre os objetos que construímos nas linguagens de programação e o banco de dados, por isso é necessário criarmos o que é chamado de Mapeamento Objeto-Relacional, onde objetos são transformados geralmente em registros em uma tabela e vice-versa.

Mapeamento Objeto Relacional

Isso surgiu porque precisamos salvar o estado de um objeto (atributos, herança, polimorfismo) em tabelas do banco de dados.

O mapeamento básico pode ser feito através de conversões manuais ou através de frameworks de persistência. Ambas as abordagens possuem vantagens e desvantagens, a conversão manual nos dá uma melhor performance enquanto que frameworks de persistência não precisamos nos preocupar com escrita de SQL.

Em Delphi temos alguns frameworks de mapeamento objeto relacional, sendo os principais o DORM (open source) e o Aurelius (comercial). Estes frameworks geralmente nos possibilitam a configuração de persistência através de arquivos XML ou o recurso de annotations ou custom atributes do Delphi.

No mapeamento objeto relacional básico o que temos é o mapeamento de classes para uma tabela do banco de dados, temos também o mapeamento de objetos da classe para registros da tabela e as propriedades da classe para colunas da tabela.

Padrão DAO (Data Access Object)

O padrão de projeto DAO surgiu com a necessidade de separarmos a lógica de negócios da lógica de persistência de dados.

Este padrão permite que possamos mudar a forma de persistência sem que isso influencie em nada na lógica de negócio, além de tornar nossas classes mais legíveis.

Classes DAO são responsáveis por trocar informações com o SGBD e fornecer operações CRUD e de pesquisas, ela deve ser capaz de buscar dados no banco e transformar em objetos ou lista de objetos através de listas genéricas, também deverá receber os objetos, converter em instruções SQL e mandar para o bando de dados. Toda interação com a base de dados se dará através destas classes, nunca das classes de negócio, muito menos de formulários.

Ao aplicarmos este padrão corretamente ele vai abstrair completamente o modo de busca e gravação dos dados, tornando isso transparente para aplicação e facilitando muito na hora de fazermos manutenção na aplicação ou migrarmos de banco de dados.

Também conseguimos centralizar a troca de dados com o SGBD (Sistema Gerenciador de Banco de Dados), teremos um ponto único de acesso a dados, tendo assim nossa aplicação um ótimo design orientado a objeto.

Autor

FILIPE DALEPIANE
filipe.dalepiane@gmail.com
Bacharel em Ciência da Computação, certificado Delphi Developer, colunista da revista Clube Delphi.
Trabalha atualmente como Analista de Sistemas na AVMB Consultoria e Assessoria em Informática em Santa Maria-RS (www.avmb.com.br).

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Estruturas de Dados em Delphi – Parte 2

Aprenda a trabalhar com estruturas de dados do tipo lista

Este artigo faz parte de um curso

Fique por Dentro

Nesta parte do curso veremos como a utilização de lista de objetos é explorada e testada através de testes unitários. São exemplificadas classes como TList e TObjectList, todas fazendo uso dos recursos prontos do Generics, que facilitam bastante o armazenamento e busca de objetos. Assim, é desnecessário descobrir qual a classe ideal, já que Generics tem a oferecer uma para cada situação. Criaremos a estrutura de dados de forma manual pra depois fazermos uso dessa biblioteca.

A partir da versão 2009 do Delphi, a Embarcadero incorporou o recurso de Generics no Delphi. Com isto foram adicionadas várias classes que facilitam o trabalho com as mais variadas estruturas de dados. Generics é um dos recursos mais importantes no Delphi, pois flexibiliza muito a escrita de código no que tange ao tratamento de lista de objetos. Como o próprio nome sugere, este recurso permite trabalhar com uma estrutura genérica em lista, o que faz toda a diferença no caso do Delphi que é uma linguagem fortemente tipada. Quando falamos que uma linguagem é fortemente tipada significa que as estruturas definidas nesta linguagem obrigatoriamente têm que ter um tipo de dado no momento de sua declaração e este tipo de dado não poderá mudar, de uma forma geral seria dizer que o que começa com o tipo inteiro, termina inteiro, por exemplo.

Na unit System.Generics.Collections temos uma relação de várias classes e records (**BOX 1**) que podem ser visualizados na **Listagem 1**.

BOX 1. Records

Os registros ou records são uma estrutura de dados muito interessante que o Delphi possui. Nela podemos definir um conjunto de campos dentro de um mesmo tipo de dados. Ao contrário de vetores e matrizes, os records podem conter tipos de dados diferentes. Da mesma maneira que os vetores e matrizes, necessitamos criar uma variável para representar os records, para então acessar seus atributos. Para podermos representar um conjunto de registros da estrutura, podemos utilizar um array de records.

Listagem 1. Classes e Records presentes na unit System.Generics.Collections

```
01 TArray = class
02 TEnumerator<T> = class abstract
03 TEnumerable<T> = class abstract
04 TArrayManager<T> = class abstract
05 TMoveArrayManager<T> = class(TArrayManager<T>)
06 TList<T> = class(TEnumerable<T>)
07 TThreadList<T> = class
08 TQueue<T> = class(TEnumerable<T>)
09 TStack<T> = class(TEnumerable<T>)
10 TPair<TKey,TValue> = record
11 TDictionary<TKey,TValue> = class(TEnumerable<TPair<TKey,TValue>>)
12 TObjectList<T: class> = class(TList<T>)
13 TObjectQueue<T: class> = class(TQueue<T>)
14 TObjectStack<T: class> = class(TStack<T>)
15 TDictionaryOwnerships = set of (doOwnsKeys, doOwnsValues);
16 TObjectDictionary<TKey,TValue> = class(TDictionary<TKey,TValue>)
17 TThreadedQueue<T> = class
```

Algoritmo e Implementação

Um algoritmo é uma sequência de passos lógicos que tem o objetivo de resolver um determinado problema. Um algoritmo diz o que deve ser feito e não como ser feito. A definição de como os passos de um algoritmo devem ser executados é a implementação do algoritmo.

Quando vamos desenvolver algoritmos computacionais devemos fazer algumas escolhas, como a linguagem de programação, paradigma de programação. Neste artigo utilizaremos a Delphi Language e o paradigma Orientado a Objetos.

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Podemos também adotar metodologias tradicionais de desenvolvimento, ou as metodologias ágeis (**BOX 2**) que estão em alta nos dias atuais.

BOX 2. Metodologias Ágeis

Metodologias tradicionais de engenharia de software, focadas em planejamento e têm sido utilizadas ao longo do tempo. O grande problema é que elas são muito burocráticas, dão ênfase em contratos fixos, documentação excessiva, vasto planejamento antecipado (prematuro). Essa abordagem não condiz mais com a realidade da maioria dos projetos na área de tecnologia, onde novos requisitos são criados, modificados ou mesmo removidos após um planejamento inicial.

Processos e metodologias modernas, como as metodologias ágeis de desenvolvimento, são evolutivas por natureza. Organizam o trabalho de forma interativa, realizado em pequenas partes, com pequenas entregas (design, código, teste, documentação), liberadas aos poucos, ao invés de um único release enorme. Dão maior ênfase na colaboração e comunicação aberta e fluente entre stakeholders, clientes e membros do time, minimizando as chances de insucesso em projetos. Os princípios básicos da abordagem ágil sugerem que indivíduos e interações são mais importantes que processos e ferramentas, trabalho deve ser feito mais focado no software em si que em uma documentação abrangente, colaboração do cliente mais que negociação de contrato e, principalmente, responder a mudanças mais que seguir um plano.

O problema é que as abordagens tradicionais de engenharia de software se baseiam em outros ramos de engenharia, como a civil e mecânica, onde os custos com planejamento são menores. Em software, estima-se que até 50% de custos são gastos com design e planejamento.

Projetos com um escopo invariável são raros, e se requisitos constantemente mudam, há um grande desperdício de esforço com análise que é descartada em virtude disso. Trabalho, tempo e dinheiro são preciosos são perdidos. Uma abordagem ágil garante que a equipe e projeto sejam capazes de se adaptar às mudanças que ocorrerão, dentro de um contexto evolutivo.

Uma metodologia ágil, como a XP (Extremme Programing), prega cinco valores principais: comunicação, feedback, simplicidade, coragem e respeito, com foco maior em disciplina do que processos. Já Scrum prega um desenvolvimento ágil baseado em iterações chamadas sprints e reuniões de acompanhamento frequentes (meetings). Aplicações reais precisam continuamente ser evoluídas, novos requisitos implementados, melhorias devem ser feitas no projeto existente (design), algumas vezes é necessário mudar de tecnologia ou plataforma, corrigir bugs, otimizar e melhorar a performance. Isso remete a uma pergunta:

O que pode ser considerado um "bom software"? Um bom software é aquele que funciona como planejado, mas depende do seu negócio. Para pequenas companhias (startups), bons softwares devem ser criados rapidamente. Para grandes corporações, o foco na construção de um software é torná-lo fácil de ser mantido ao longo do tempo.

Nesse sentido, é necessário tomar cuidado com bons princípios de design, como criar componentes com baixo acoplamento, reutilizáveis, que garantam que alterações em componentes não afetam outros, que objetos tenham responsabilidades únicas, separadas, partes facilmente substituíveis, complexidade encapsulada. Esses são princípios básicos da programação orientada a objetos que, vistos dentro de um contexto de uma abordagem ágil evolutiva, vão tornar o sistema de software mais fácil de testar, evoluir, manter e entender.

Classe TFuncionario

Partiremos do princípio de desenvolvimento de uma lista de funcionários. Na **Listagem 2** temos toda a unit uFuncionario contendo a classe TFuncionario. Esta classe possui duas propriedades públicas chamadas Nome e Salário, informações estas abstraídas de um cadastro simples de funcionários.

Para cada uma das propriedades temos um atributo privado, FNome e FSalario, que servem para armazenar em memória estas informações, pois como já é sabido, as propriedades em Delphi são apenas um mecanismo de controle e encapsulamento de informações, são os atributos que guardam as informações propriamente ditas.

Listagem 2. Classe TFuncionario

```
01 unit uFuncionario;
02 interface
03 type
04  TFuncionario = class(TObject)
05  private
06   FNome: string;
07   FSalario: Double;
08   function GetNome: string;
09   procedure SetNome(const Value: string);
10   function GetSalario: Double;
11   procedure SetSalario(const Value: Double);
12  public
13   function ToString: string; override;
14   function Equals(Obj: TObject): Boolean; override;
15   property Nome: string read GetNome write SetNome;
16   property Salario: Double read GetSalario write SetSalario;
17  end;
18 implementation
19 uses
20  System.SysUtils;
21 { TFuncionario }
22 function TFuncionario.Equals(Obj: TObject): Boolean;
23 var
24  LFunc: TFuncionario;
25 begin
26  LFunc := TFuncionario(Obj);
27  result := Self.FNome = LFunc.Nome;
28 end;
29 function TFuncionario.GetNome: string;
30 begin
31  result := Self.FNome;
32 end;
33 function TFuncionario.GetSalario: Double;
34 begin
35  result := Self.FSalario;
36 end;
37 procedure TFuncionario.SetNome(const Value: string);
38 begin
39  Self.FNome := Value;
40 end;
41 procedure TFuncionario.SetSalario(const Value: Double);
42 begin
43  Self.FSalario := Value;
44 end;
45 function TFuncionario.ToString: string;
46 begin
47  result := Self.Nome + ': ' + FloatToStr(Self.Salario);
48 end;
49 end.
```

Para cada propriedade, deve-se haver um mecanismo de leitura e escrita para o atributo. Esta leitura e escrita podem ser feitas diretamente em cima do atributo, ou como no exemplo, cria-se métodos de leitura e escrita para cada propriedade, de maneira que podemos posteriormente controlar e validar de uma maneira mais eficaz a leitura e escrita nesses atributos.

A classe TObject nas últimas versões do Delphi incorporou dois novos métodos, o ToString e o Equals. O ToString serve para que possamos fornecer uma descrição para o objeto criado, geralmente usamos esta informação para vermos o resultado do objeto. Por padrão este método retorna o nome da classe instanciada,

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

conforme podemos ver na **Listagem 3**. Em nosso exemplo trazemos o nome do funcionário mais o seu salário, para que possamos visualizar estas informações em tela. Já o método Equals serve para fornecermos uma maneira de compararmos dois objetos e verificarmos se são iguais. Por padrão, o Delphi compara as duas referências para verificar se são iguais, conforme podemos verificar a partir da linha 6, mas no nosso exemplo utilizamos o nome do funcionário para efetuar a comparação.

Listagem 3. ToString e Equals da classe base TObject

```
01 function TObject.ToString: string;
02 begin
03   Result := ClassName;
04 end;
05
06 function TObject.Equals(Obj: TObject): Boolean;
07 begin
08   Result := Obj = Self;
09 end;
```

Para facilitar os testes foi criada uma classe chamada TTeste (**Listagem 4**), onde para cada teste efetuado cria-se uma procedure nova. O primeiro teste é o método Teste1, onde vemos a partir da linha 12 a declaração de uma referência para a classe TFuncionario. Posteriormente é criada uma instância para ela, chamada Funcionario. Então em um bloco try/finally setarmos as propriedades do objeto são setadas e o método ToString é executado, onde é possível verificar se se tudo ocorreu bem.

No arquivo de projeto (.dpr), basta que chamemos cada método da classe TTeste, observando que não é preciso criar uma instância da classe TTeste, pois os métodos nela contidos são métodos de classe e não métodos de objeto. Na orientação a objetos costumamos chamar estes métodos de métodos estáticos, que não precisam de uma instância para serem chamados.

Classe TListaFuncionario

Para criarmos uma estrutura de dados de lista de funcionários, partiremos do princípio da criação da classe TListaFuncionario. Nossa lista deve atender os seguintes pontos:

- Adicionar um funcionário na lista;
- Buscar um funcionário a partir de uma posição;
- Remover um funcionário de uma posição;
- Verificar se um funcionário existe na lista;
- Informar o número de funcionários existentes.

Nossos dados serão armazenados internamente nesta lista em um array dinâmico, que deverá ser redimensionado a cada novo item incluído. Cada posição deste array não irá armazenar um funcionário, mas sim uma referência para um objeto da classe TFuncionario na memória.

Os arrays dinâmicos são aqueles cujo tamanho não é informado na sua declaração, devendo o desenvolvedor definir essa capacidade em tempo de execução. A declaração de arrays dinâmicos é semelhante à dos arrays estáticos, apenas não é informado o intervalo de dados que esta estrutura irá suportar, devendo o desenvolvedor informar isso em tempo de execução através da função SetLength e redimensionando quando necessário o array através da função VarArrayRedim. Para utilizar ambas as funções, não é necessária nenhuma declaração de unit, já que ambas pertencem a unit System, que é a mais básica do Delphi e é importada implicitamente em todos os projetos. Todos os Arrays dinâmicos definidos através da função SetLength iniciam na posição 0, sem exceção, sendo eles vetores ou matrizes.

Listagem 4. Classe com Métodos de Teste

```
01 unit uTeste;
02 interface
03 type
04  TTeste = class
05    class procedure Teste1;
06    class procedure Teste2;
07    class procedure Teste3;
08  end;
09 implementation
10 { TTeste }
11 uses uListaFuncionario, uFuncionario;
12 class procedure TTeste.Teste1;
13 var
14  Funcionario: TFuncionario;
15 begin
16  Funcionario := TFuncionario.Create;
17  try
18    Funcionario.Nome := 'Maria Santos';
19    Funcionario.Salario := 1520.25;
20    Writeln(Funcionario.ToString);
21  finally
22    Funcionario.Free;
23  end;
24 end;
25 class procedure TTeste.Teste2;
26 var
27  Func1, Func2, Func3: TFuncionario;
28 begin
29  Func1 := TFuncionario.Create;
30  Func2 := TFuncionario.Create;
31  Func3 := TFuncionario.Create;
32  try
33    Func1.Nome := 'Maria Santos';
34    Func2.Nome := 'Pedro Silva';
35    Func3.Nome := 'Maria Santos';
36    if Func1.Equals(Func2) then
37      Writeln('Func1 igual a Func2');
38    if Func1.Equals(Func3) then
39      Writeln('Func1 igual a Func3');
40    if Func2.Equals(Func3) then
41      Writeln('Func2 igual a Func3');
42  finally
43    Func1.Free;
44    Func2.Free;
45    Func3.Free;
46  end;
47 end;
```

Na **Listagem 5** temos a definição da interface da classe TListaFuncionario, depois veremos cada método individualmente no decorrer do artigo.

Podemos observar que possuímos dois métodos Adiciona, onde fazemos um overload (**BOX 3**), mudando apenas seus parâmetros.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

O primeiro método a ser analisado é o Adiciona da classe TListaFuncionario, nele recebemos um objeto do tipo TFuncionario como parâmetro. O objetivo deste método é receber um objeto de TFuncionario e armazená-lo em uma posição livre do array de funcionários definido na seção private.

Para percorrer o vetor faz-se um for do menor ao maior índice do vetor de funcionários. Logo após, para cada posição é testado se aquela posição do vetor já possui alguma referência, se não tiver, ou seja, o vetor[i] for igual a nil, podemos fazer a atribuição da referência deste objeto naquela posição, para depois chamarmos o break para encerrar a iteração do laço, já que encontramos uma posição vaga e não são mais necessárias iterações.

A primeira versão do método Adiciona pode ser vista na sua totalidade na **Listagem 6** logo a seguir.

BOX 3. Overload

Overload de Métodos são dois ou mais métodos com o mesmo nome, porém número ou tipos de dados dos parâmetros diferentes, o que faz com que o compilador saiba qual método chamar na hora da execução. Um exemplo é o método GetConnection da classe TDBXConnectionFactory, ele possui duas implementações:

```
GetConnection(const ConnectionName: string; const UserName: string;
const Password: string): TDBXConnection;
GetConnection(ConnectionProperties: TDBXProperties): TDBXConnection;
```

O Overload de métodos serve para que possamos ter em nossas classes dois métodos com o mesmo nome. Isso serve também para métodos construtores. Para que o compilador possa diferenciar qual método chamar, os métodos de mesmo nome devem ter um número de parâmetros diferentes ou o mesmo número de parâmetros iguais e apenas os tipos de dados recebidos por ele diferentes. Devemos adicionar a diretiva overload em cada método. Veja nos exemplos a seguir, uma classe com estes métodos iria compilar sem problemas:

- procedure Teste(Value: Integer); overload;
- procedure Teste(Value: Integer; Str: String) overload;
- procedure Teste(Value: String); overload;

Listagem 5. Interface da classe TListaFuncionario

```
01 TListaFuncionario = class
02 private
03   Funcionarios: array of TFuncionario;
04 public
05   procedure Adiciona(AFuncionario: TFuncionario); overload;
06   procedure Adiciona(AFuncionario: TFuncionario; APosicao: Integer);
       overload;
07   function Busca(APosicao: Integer): TFuncionario;
08   procedure Remove(APosicao: Integer);
09   function Existe(AFuncionario: TFuncionario): Boolean;
10   function Tamanho: Integer;
11   function ToString: string; override;
12   constructor Create;
13 end;
```

Para testarmos o método Adiciona declaramos uma referência chamada ListaFuncionarios para a classe TListaFuncionario e também duas referências chamadas Funcionario1 e Funcionario2 para a classe TFuncionario.

Como vemos na **Listagem 7**, criamos os objetos das classes referenciadas e setamos as propriedades dos funcionarios. Por fim adicionamos as duas referências na lista e podemos visualizar seus dados através da chamada do método ToString.

Listagem 6. Método Adiciona da Classe TListaFuncionario

```
01 procedure TListaFuncionario.Adiciona(AFuncionario: TFuncionario);
02 var
03   i: Integer;
04 begin
05   for i := Low(Funcionarios) to High(Funcionarios) do
06     if Funcionarios[i] = nil then
07     begin
08       Funcionarios[i] := AFuncionario;
09       Break;
10     end;
11 end;
```

Listagem 7. Método Adiciona da classe TListaFuncionario

```
01 class procedure TTeste.Teste3;
02 var
03   ListaFuncionarios: TListaFuncionario;
04   Funcionario1, Funcionario2: TFuncionario;
05 begin
06   ListaFuncionarios := TListaFuncionario.Create;
07   Funcionario1 := TFuncionario.Create;
08   Funcionario2 := TFuncionario.Create;
09   try
10     Funcionario1.Nome := 'João da Silva';
11     Funcionario1.Salario := 1250.25;
12     Funcionario2.Nome := 'Maria dos Santos';
13     Funcionario2.Salario := 2500.00;
14     ListaFuncionarios.Adiciona(Funcionario1);
15     ListaFuncionarios.Adiciona(Funcionario2);
16     Writeln(ListaFuncionarios.ToString);
17   finally
18     Funcionario1.Free;
19     Funcionario2.Free;
20     ListaFuncionarios.Free;
21   end;
22 end;
```

Para facilitar a visualização dos dados em tela, sobrescrevemos também na classe TListaFuncionario o método ToString **(Listagem 8).** Nele percorremos todos os itens do array de funcionários, verificando primeiramente se este não aponta para um elemento inválido de memória e depois concatenando o retorno do método ToString da classe funcionário que codificamos anteriormente mais uma quebra de linha para melhor organização dos dados em tela.

O método Tamanho serve para retornar o total de itens presentes na nossa lista de funcionários, para isso é inicializado o resulta, uma boa prática de programação, já que o Delphi não inicializa variáveis do tipo Integer, de maneira que caso ocorra algum erro durante o restante do processamento, garantimos que o result será igual a 0.

Dentro de um laço que vai do início até o final do vetor, verificamos se existe uma referência válida na posição de memória. Se sim, incrementa-se 1 na variável Aux, para no fim retornarmos para quem chamou este método o número total de objetos de

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

TFuncionario existem no vetor de funcionários. Podemos ver a implementação na **Listagem 9.**

Listagem 8. Método ToString da Classe TListaFuncionario

```
01 function TListaFuncionario.ToString: string;
02 var
03   i: Integer;
04 begin
05   for i := Low(Funcionarios) to High(Funcionarios) do
06     if Funcionarios[i] <> nil then
07       result := Result + Funcionarios[i].ToString + sLineBreak;
08 end;
```

Listagem 9. Método Tamanho da classe TListaFuncionario

```
01 function TListaFuncionario.Tamanho: Integer;
02 var
03   Aux: integer;
04   i: Integer;
05 begin
06   Result := 0;
07   i := 0;
08   for i := Low(Funcionarios) to High(Funcionarios) do
09     if Assigned(Funcionarios[i]) then
10       Aux := Aux + 1;
11   Result := Aux;
12 end;
```

O método Existe utiliza também um laço de repetição que irá percorrer todo o vetor de funcionários que está internamente na classe TListaFuncionario. Para verificarmos e o objeto daquela posição do vetor é igual a referência do objeto passado por memória, fazemos uma simples comparação de igualdade. Aqui não estamos comparando o conteúdo daquele objeto funcionário, como o nome e o salário, e sim a referência de memória, se ambos estão apontando para as mesmas posições de memória.

No momento em que a igualdade for verdadeira utilizamos o método Exit, que a partir das versões mais novas do Delphi além de sair fora do procedimento, também permite que no caso de estarmos o adicionando em uma função, devolver um argumento para que o chamou, da mesma maneira que faz o result.

O método Existe, contido na classe TListaFuncionario, pode ser visto na sua totalidade na **Listagem 10**.

O método busca é bastante simples, ele deve apenas acessar determinada posição do vetor de referências a funcionários e devolver o objeto solicitado, conforme pode ser visto na **Listagem 11**.

Porém, devemos ter o cuidado de testar se existe uma referência de memória válida para a posição solicitada por quem chamou o método. Por isso, utilizamos a função Assigned que retorna True caso exista uma referência válida para aquele objeto desejado.

No segundo método de Adição de funcionário na lista, temos já definida a posição em que este deve ser inserido. Neste caso é preciso ter um cuidado para que a posição que está se desejando fazer a adição do item é válida. Por esse motivo utilizamos novamente a função Assigned em conjunto com o operador lógico Not, verificando se aquela posição de memória está livre, ou seja, não está ocupada.

Caso a posição de memória esteja já ocupada lança-se uma exceção informando que aquela posição é inválida. Vemos a implementação na **Listagem 12**

Listagem 10. Método Existe da classe TListaFuncionario

```
01 function TListaFuncionario.Existe(AFuncionario: TFuncionario): Boolean;
02 var
03   Ind: Integer;
04 begin
05   Result := False;
06   for Ind := Low(Self.Funcionarios) to High(Self.Funcionarios) do
07   begin
08     if Self.Funcionarios[Ind] = AFuncionario then
09       Exit(True);
10   end;
11 end;
```

Listagem 11. Método Busca da Classe TListaFuncionario

```
01 function TListaFuncionario.Busca(APosicao: Integer): TFuncionario;
02 var
03   LFuncionario: TFuncionario;
04 begin
05   LFuncionario := nil;
06   if Assigned(Self.Funcionarios[APosicao]) then
07     LFuncionario := Self.Funcionarios[APosicao];
08   Result := LFuncionario;
09 end;
```

Listagem 12. Método Adiciona com posição na lista definida

```
01 procedure TListaFuncionario.Adiciona
     (AFuncionario: TFuncionario; APosicao: Integer);
02 begin
03   if not Assigned(Self.Funcionarios[APosicao]) then
04     Self.Funcionarios[APosicao] := AFuncionario
05   else raise Exception.Create('Posição Inválida!');
06 end;
```

O método analisado a seguir é o Remover: nele primeiramente verificamos se a posição passada como parâmetro é nula, caso seja é lançada uma exceção do tipo EArgumentException, a mais indicada de lançamos quando recebemos um parâmetro que é inválido. Isso facilita na hora de analisarmos as exceções geradas por um sistema, mas às vezes nos acostumamos a sempre chamar a classe base de todas as exceções a Exception, devemos procurar lançar as exceções corretas ou criar nossas próprias classes de exceções.

Porém somente isto não basta, não devemos apenas atribuir nulo a posição de memória passada por parâmetro, se assim fizermos ficaremos com uma posição vaga no meio da lista, e isso não é desejado, portanto o que fazemos é mudarmos o índice de todas os objetos a partir da posição passada por parâmetro, assim ficamos sem nenhuma lacuna no vetor, conforme **Listagem 13**.

Para renomearmos classes, atributos e objetos o Delphi e a maioria das IDEs modernas nos oferecem o poderoso recurso de Refactorings (**BOX 4**).

Classe TList<>

A classe TList<> nos oferece diversos métodos que facilitam bastante a vida do desenvolvedor na hora de trabalhar com listas.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Para usá-la deve-se declarar a unit System.Generics.Collections. A primeira e mais simples é a função Add.

Como podemos observar na **Listagem 14**, no exemplo criaremos uma lista de strings e fazemos a adição de três nomes e depois o imprimimos na tela para observarmos o resultado final, a partir da linha 18.

O segundo método que iremos analisar da classe TList<T> é o AddRange. Com ele é possível adicionar mais de um item simultaneamente no final da lista. Caso seja necessário, a capacidade da lista é aumentada.

BOX 4. Refactorings

Refatoração é o processo de modificar um sistema de software sem alterar o comportamento externo do código, melhorando sua estrutura interna. É uma forma disciplinada de limpar minimizando as chances de introduzir erros. Quando se refatora está se melhorando o design do código depois de ele ter sido feito. Primeiro um bom projeto deveria ser feito, a seguir a sua codificação, mas nem sempre é assim que as coisas acontecem.

Com a evolução do sistema de software, mudanças ocorrerão, e a sua estrutura e integridade original diminuem. Dessa forma, essas transformações, no caso as refatorações, podem melhorar a estrutura do projeto. Embora a reestruturação crie novas versões que implementam ou propõem mudanças no objetivo final do sistema, normalmente não envolvem modificações referentes a novos requisitos, ou seja, não visam implementar novas funcionalidades no sistema de software. No entanto, podem conduzir a observar melhor o objetivo do sistema, já que sugerem mudanças que poderem melhorar sua estrutura interna.

Listagem 13. Método Remover da classe TListaFuncionario

```
01 procedure TListaFuncionario.Remove(APosicao: Integer);
02 var
03   i: Integer;
04 begin
05   if not Assigned(Self.Funcionarios[APosicao]) then
06     raise EArgumentException.Create('Posição informada inválida!');
07   for i := APosicao to High(Self.Funcionarios) -1 do
08     Self.Funcionarios[i] := Self.Funcionarios[i + 1];
09 end;
```

Listagem 14. Método Add da classe TList<T>

```
01 class procedure TTeste.Teste9;
02 var
03   ListaNomes: TList<string>;
04   Aux: string;
05 begin
06   ListaNomes := TList<string>.Create;
07   try
08     ListaNomes.Add('João Silva');
09     ListaNomes.Add('Maria Santos');
10     ListaNomes.Add('José Lima');
11     for Aux in ListaNomes do
12       Writeln(Aux);
13   finally
14     ListaNomes.Free;
15   end;
16 end;
17
18 João Silva
19 Maria Santos
20 José Lima
```

A classe TList<T> possui o evento OnNotify, que deve ser configurado para sermos notificados da adição de cada item na lista. No caso do AddRange, ao adicionarmos dois itens simultâneos, este evento é disparado duas vezes. Na **Listagem 15** vemos que foram adicionados dois itens no final da lista.

Para efetuarmos a busca por determinado item dentro da lista e verificarmos se ele está lá presente, uma das opções é a utilização do método Contains. Este método retornará True se o elemento estiver na lista e False se não existir. Porém este método não nos devolve a posição do item onde foi encontrado o item passado, ao contrário do BinarySearch, que será visto na **Listagem 16**.

Listagem 15. Método AddRange da classe TList<T>

```
01 class procedure TTeste.Teste11;
02 var
03   ListaNomes: TList<string>;
04   Aux: string;
05 begin
06   ListaNomes := TList<string>.Create;
07   try
08     ListaNomes.Add('Maria Santos');
09     ListaNomes.Add('José Lima');
10     ListaNomes.AddRange(['Lucia Gomes', 'Antonio Machado']);
11     for Aux in ListaNomes do
12       Writeln(Aux);
13   finally
14     ListaNomes.Free;
15   end;
16 end;
17
18 Maria Santos
19 José Lima
20 Lucia Gomes
21 Antonio Machado
```

Listagem 16. Procurando item através do método Contains da classe TList<T>

```
01 class procedure TTeste.Teste13;
02 var
03   ListaNomes: TList<string>;
04   Aux: integer;
05   Nome: string;
06 begin
07   ListaNomes := TList<string>.Create;
08   try
09     ReadIn(Nome);
10     ListaNomes.Add('Maria Santos');
11     ListaNomes.Add('José Lima');
12     ListaNomes.AddRange(['Lucia Gomes', 'Antonio Machado']);
13     if ListaNomes.Contains(Nome) then
14       Writeln('Elemento encontrado')
15     else Writeln('Elemento não encontrado');
16   finally
17     ListaNomes.Free;
18   end;
19 end;
```

O método que usamos para buscar a posição de determinado item dentro de uma lista é o IndexOf (**Listagem 17**). Nele passamos o parâmetro Value, que é o dado que se deseja procurar na lista. Caso não seja encontrado nenhum item é retornado -1, senão é retornado o índice da posição na lista, lembrando que os índices de uma lista começam sempre na posição 0.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

A busca efetuada tanto pelo Contains, quanto pelo IndexOf é Linear, ou seja, percorre a lista até encontrar o valor passado como parâmetro, de forma sequencial, retornando assim o índice do valor encontrado. A Complexidade da busca linear é O(n).

Listagem 17. Buscando a posição de um item através do método IndexOf

```
01 class procedure TTeste.Teste14;
02 var
03  ListaNomes: TList<string>;
04  Aux: integer;
05  Nome: string;
06 begin
07  ListaNomes := TList<string>.Create;
08  try
09   ReadIn(Nome);
10   ListaNomes.Add('Maria Santos');
11   ListaNomes.Add('José Lima');
12   ListaNomes.AddRange(['Lucia Gomes','Antonio Machado']);
13   Aux := ListaNomes.IndexOf(Nome);
14   if Aux > 0 then
15    WriteIn('Elemento encontrado na posição ' + IntToStr(Aux))
16   else WriteIn('Elemento não encontrado');
17  finally
18   ListaNomes.Free;
19  end;
20 end;
```

O próximo método da classe TList<T> a ser analisado é o BinarySearch. A pesquisa ou busca binária é um algoritmo de busca em vetores e segue o mecanismo de divisão e conquista. Ela parte do pressuposto de que o vetor está ordenado e realiza sucessivas divisões do espaço de busca comparando o elemento buscado (chave) com o elemento no meio do vetor. Se o elemento do meio do vetor for a chave, a busca termina com sucesso. Caso contrário, se o elemento do meio vier antes do elemento buscado, então a busca continua na metade posterior do vetor. E finalmente, se o elemento do meio vier depois da chave, a busca continua na metade anterior do vetor.

O método BinarySearch recebe o item que deve ser pesquisado por parâmetro e uma variável de saída que será preenchida caso o item seja encontrado na lista. O retorno desta função é do tipo Boolean e informa se foi encontrado ou não um item dentro da lista.

Como vemos na **Listagem 18**, criamos uma variável para ser informada ao executar o procedimento e mostrará se foi ou não encontrado este elemento.

O método BinarySearch na maioria das vezes é mais rápido que o IndexOf, isto porque a busca binária geralmente é mais rápida que a busca linear, porém ela exige que a lista esteja ordenada para potencializar a busca.

Os dois primeiros métodos que vistos a seguir (**Listagem 19**) servem para coletarmos informações da nossa lista. O Capacity serve para nos mostrar a capacidade de itens que pode conter nossa lista, lembrando que os métodos Add e AddRange redimensionam esta lista afim de poder inserir mais dados.

Por último, mas não menos importante, temos o método Clear, que serve para limpar os itens da nossa lista.

Listagem 18. Método BinarySearch da classe TList<T>

```
01 class procedure TTeste.Teste12;
02 var
03  ListaNomes: TList<string>;
04  Aux: integer;
05  Nome: string;
06 begin
07  ListaNomes := TList<string>.Create;
08  try
09   ReadIn(Nome);
10   ListaNomes.Add('Maria Santos');
11   ListaNomes.Add('José Lima');
12   ListaNomes.AddRange(['Lucia Gomes','Antonio Machado']);
13   if ListaNomes.BinarySearch(Nome, Aux) then
14    WriteIn('Elemento encontrado na posição ' + IntToStr(Aux))
15   else WriteIn('Elemento não encontrado');
16  finally
17   ListaNomes.Free;
18  end;
19 end;
```

Listagem 19. Métodos Capacity, Count e Clear

```
01 class procedure TTeste.Teste15;
02 var
03  ListaNomes: TList<string>;
04 begin
05  ListaNomes := TList<string>.Create;
06  try
07   ListaNomes.Add('Maria Santos');
08   ListaNomes.Add('José Lima');
09   ListaNomes.AddRange(['Lucia Gomes','Antonio Machado']);
10   writeln(inttostr(ListaNomes.Capacity));
11   writeln(inttostr(ListaNomes.Count));
12   ListaNomes.Clear;
13  finally
14   ListaNomes.Free;
15  end;
16 end;
```

Os dois métodos vistos a seguir na **Listagem 20** servem para removermos itens da nossa lista. O método Delete recebe um AIndex como parâmetro, este AIndex é o índice do item que desejamos remover da lista. Já o DeleteRange, passamos além do AIndex, devemos passar também o ACount, que é a quantidade total de itens que deseja se remover a partir do AIndex.

Na **Listagem 21** temos a utilização do método Extract. Devemos passar um dado que desejamos extrair da lista, então ele nos retornará no retorno do método Extract e será removido dos itens da lista, não podendo mais ser acessado.

O método analisado a seguir é o Exchange, que serve para trocarmos dois itens de posição dentro da lista. Devemos passar os dois índices (Index1, Index2), os quais devem ser trocados de posições na lista.

No final da **Listagem 22** podemos observar a lista dos itens após feita a troca de posições, vendo como funcionou perfeitamente a troca.

Os dois métodos a seguir (**Listagem 23**) são bastante simples e somente nos retornam o valor do primeiro e último item da lista. Observe que o retorno das funções First e Last é do tipo String porque definimos na sua declaração que esta lista conteria apenas dados deste tipo. Se tivéssemos uma lista de Inteiros ou Reais, seria necessária uma conversão na função Writeln.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

No método Insert da classe TList, já devemos especificar com antecedência em que posição da lista queremos adicionar aquele item, ao contrário do Add e AddRange, que adicionam os itens sempre no final da lista. Como pode ser visto na **Listagem 24**, o item Willian Silveira foi adicionado na posição 2 da lista.

Classe TObjectList<T>

A classe TObjectList<T> é bastante semelhante a classe TList<T>, ela possui todas as suas características, com a diferença de ser uma classe para armazenar somente listas de objetos, não podem ser adicionados nenhum tipo primitivo como strings, inteiros, números reais, etc.

Ela possui uma grande vantagem em relação à TList<T>, pois quando trabalhamos com a TList<T> armazenando objetos, no final da sua utilização, além de liberar a lista da memória, precisamos liberar todos os objetos nela contidas. É natural entender o porquê desta limitação da classe TList<T>, isto ocorre porque ela deve trabalhar tanto com tipos primitivos como com referências de objetos, de maneira que precisa ser genérica e atender a esses dois tipos muito diferentes de informações.

Listagem 20. Métodos Delete e DeleteRange da classe TList<T>

```
01 class procedure TTeste.Teste16;
02 var
03   ListaNomes: TList<string>;
04   Aux: string;
05 begin
06   ListaNomes := TList<string>.Create;
07   try
08     ListaNomes.Add('Maria Santos');
09     ListaNomes.Add('José Lima');
10     ListaNomes.Add('Pedro Silva');
11     ListaNomes.AddRange(['Lucia Gomes', 'Antonio Machado']);
12     ListaNomes.Delete(1);
13     ListaNomes.DeleteRange(1, 2);
14     for Aux in ListaNomes do
15       Writeln(Aux)
16   finally
17     ListaNomes.Free;
18   end;
19 end;
```

Listagem 21. Método Extract da classe TList<T>

```
01 class procedure TTeste.Teste17;
02 var
03   ListaNomes: TList<string>;
04   Aux: string;
05 begin
06   ListaNomes := TList<string>.Create;
07   try
08     ListaNomes.Add('Maria Santos');
09     ListaNomes.Add('José Lima');
10     ListaNomes.Add('Pedro Silva');
11     ListaNomes.AddRange(['Lucia Gomes', 'Antonio Machado']);
12
13     Writeln(ListaNomes.Extract('Pedro Silva'));
14
15     for Aux in ListaNomes do
16       Writeln(Aux)
17   finally
18     ListaNomes.Free;
19   end;
20 end;
```

Listagem 22. Trocando itens através do método Exchange

```
01 class procedure TTeste.Teste18;
02 var
03   ListaNomes: TList<string>;
04   Aux: string;
05 begin
06   ListaNomes := TList<string>.Create;
07   try
08     ListaNomes.Add('Maria Santos');
09     ListaNomes.Add('José Lima');
10     ListaNomes.Add('Pedro Silva');
11     ListaNomes.AddRange(['Lucia Gomes', 'Antonio Machado']);
12
13     ListaNomes.Exchange(1, 3);
14
15     for Aux in ListaNomes do
16       Writeln(Aux)
17   finally
18     ListaNomes.Free;
19   end;
20 end;
21
22 Maria Santos
23 Lucia Gomes
24 Pedro Silva
25 José Lima
26 Antonio Machado
```

Listagem 23. Pegando primeiro e último item da lista

```
01 class procedure TTeste.Teste18;
02 var
03   ListaNomes: TList<string>;
04   Aux: string;
05 begin
06   ListaNomes := TList<string>.Create;
07   try
08     ListaNomes.Add('Maria Santos');
09     ListaNomes.Add('José Lima');
10     ListaNomes.Add('Pedro Silva');
11
12     Writeln(ListaNomes.First);
13     Writeln(ListaNomes.Last);
14   finally
15     ListaNomes.Free;
16   end;
17 end;
```

Listagem 24. Utilizando método Insert da classe TList<T>

```
01 class procedure TTeste.Teste18;
02 var
03   ListaNomes: TList<string>;
04   Aux: string;
05 begin
06   ListaNomes := TList<string>.Create;
07   try
08     ListaNomes.Add('Maria Santos');
09     ListaNomes.Add('José Lima');
10     ListaNomes.Add('Pedro Silva');
11
12     ListaNomes.Insert(1, 'Willian Silveira');
13
14     for Aux in ListaNomes do
15       Writeln(Aux)
16   finally
17     ListaNomes.Free;
18   end;
19 end;
20
21 Maria Santos
22 Willian Silveira
23 José Lima
24 Pedro Silva
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 25. Utilização da Classe TObjectList<T> para Listas de Objetos

```
01 class procedure TTeste.Teste10;
02 var
03  ListaObjetos: TObjectList<TFuncionario>;
04  Func1, Func2, Func3, FuncAux: TFuncionario;
05 begin
06  ListaObjetos := TObjectList<TFuncionario>.Create();
07  Func1 := TFuncionario.Create;
08  Func2 := TFuncionario.Create;
09  Func3 := TFuncionario.Create;
10  try
11   Func1.Nome := 'Maria Santos';
12   Func1.Salario := 1525.00;
13   Func2.Nome := 'Pedro Silva';
14   Func2.Salario := 1250.25;
15   Func3.Nome := 'João Lima';
16   Func3.Salario := 2750.50;
17   ListaObjetos.Add(Func1);
18   ListaObjetos.Add(Func2);
19   ListaObjetos.Add(Func3);
20   for FuncAux in ListaObjetos do
21   begin
22    Writeln(FuncAux.ToString);
23   end;
24 finally
25   ListaObjetos.Free;
26 end;
27 end;
```

Já na classe TobjectList<T>, ao final do processo precisamos apenas liberar o objeto que representa a lista de objetos, que automaticamente todos os itens contidos no interior da estrutura de dados são desalocados e não precisamos nos preocupar com essa tarefa.

Como a estrutura de dados possui apenas objetos, é natural que ela possua características que facilitem o trabalho com objetos. Esta vantagem facilita, pois não corremos o risco de ficarmos com vazamentos de memórias e referências inválidas. Um código como o contido na **Listagem 25** irá funcionar perfeitamente e toda a memória alocada para os objetos será liberada sem nenhum problema.

A estrutura de dados do tipo lista é com certeza a mais utilizada pelos analistas e desenvolvedores que utilizam o paradigma orientado a objetos. Esta estrutura aliada ao recurso de Generics é um poderoso recurso que toda a linguagem de programação orientada a objetos moderna deve oferecer. Não é à toa foi um dos primeiros recursos adicionados pela Embarcadero quando assumiu o Delphi.

Autor

FILIPE DALEPIANE
filipe.dalepiane@gmail.com
Bacharel em Ciência da Computação, certificado Delphi Developer, colunista da revista Clube Delphi.
Trabalha atualmente como Analista de Sistemas na AVMB Consultoria e Assessoria em Informática em Santa Maria-RS (www.avmb.com.br).

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

DEVMEDIA

DÊ UM SALTO EM CONHECIMENTO!

Acesse o maior portal para desenvolvedores da América Latina!

20 mil posts

430 mil cadastrados

10 milhões de page-views por mês

http://www.devmedia.com.br/

DEVMEDIA

Somos tão apaixonados por tecnologia que o nome da empresa

diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,

assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única. Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486